Iterative solution of nonsymmetric linear systems arising from process modelling applications

Brooking, Christopher George

Award date:
1997

Awarding institution:
University of Bath

Iterative solution of nonsymmetric linear systems arising from process modelling applications

submitted by

Christopher George Brooking

for the degree of Ph.D.

of the

University of Bath

1997

ProQuest LLC
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106-1346

Summary

In this thesis, we consider the iterative solution of a sequence of nonsymmetric linear systems, arising from Newton's method in the time-integration of systems of differential and algebraic equations (DAE's). These DAE's are generated by SPEEDUP, a process modelling tool produced by Aspentech. The linear systems are large and sparse, and are severly ill-conditioned. To solve the linear systems, Krylov subspace methods are considered, primarily the GMRES method introduced by Saad and Schultz in 1986.

We use a novel approach to the GMRES error analysis to obtain convergence rate estimates for matrices whose spectra are clustered with the possibility of small numbers of outlying eigenvalues. Preconditioning is required to complete the solution of the linear systems, so the popular incomplete factorisation methods are tried. Pseudospectral methods are used in conjunction with these estimates to explain the poor performance of such preconditioners on several test matrices. An alternative preconditioner based on a full factorisation of a previous matrix from the sequence is shown to produce the desired convergence behavior.

This preconditioner is used to derive a linear solver for use in SPEEDUP integration runs. The robustness of this solver is proven on several 'real-world' SPEEDUP problems arising from the process modelling industry. The performance of this solver is compared with that of several direct methods, and is found to be faster when used with a standard Newton solver, but slower when compared with the optimal combination of direct solver and approximate Newton solver. The performance of inexact Newton methods is investigated, and results show that any performance gain from using these methods is negligible. Finally, alternative iterative solvers to GMRES are used, and their poor relative performance is explained.

I would like to thank, in no particular order, the following people for their various contributions to this thesis, and life in general over the last few years:

- Alastair Spence, my supervisor, for his many helpful suggestions and ideas regarding the project, and in spite of the fact that he can't spell 'George'.
- Tony Garratt, my industrial supervisor at Apentech UK, for his time and patience helping me with SPEEDUP teething problems, and his enthusiasm for the project. Also at ATUK, Peter Ward for his support of the project, and Bryan Davidson for his invaluable help with interfacing my code with that of SPEEDUP (and his job).
- My parents, for their emotional and financial support during my undergraduate years, without which I wouldn't have got this far.
- Alastair Fitt, for starting the ball rolling with MA212 and my final year project, and Ron King, my undergraduate tutor, for helpful career guidance in my third year at Southampton.
- Michelle Orme, officemate from the very start to the bitter end!
- Steve 'Beavis' Benbow, for Bass and Madras, and the odd mathematical discussion.
- Stan and Nic, for helping me get that BUSA medal after 6 years at University.
- Peta, for putting up with me for all this time.
- The EPSRC for their support and funding of this project.

And the end and the beginning were always there
Before the beginning and after the end.
And all is always now.

T.S. Eliot

Contents

Chapter 1

Introduction

1.1 A sequence of linear equations

This thesis is concerned with the iterative solution of sequences of large, sparse, non-symmetric linear systems

$$J_l x = b_l, \quad l = 1, 2, \ldots \tag{1.1}$$

where $J_l \in \mathbb{R}^{n \times n}$ and x, $b_l \in \mathbb{R}^n$, arising from SPEEDUP [52], a process modelling software tool. SPEEDUP is a software package produced by AspenTech for use by the process engineering industry. In real-world problems, the matrix dimension n can be as large as 30,000.

The matrices J_l arise from time integration of nonlinear systems of differential and algebraic equations (DAE's), which we will briefly review later. The current solution method for (1.1) is to use a direct method based on sparse LU factorisation, which can result in poor performance when many additional factorisations are required during an integration, as is the case when sudden changes happen in the physical processes being modelled. This is the motivation for investigating iterative linear solvers, namely modern Krylov subspace methods such as GMRES [61]. Unlike matrices arising from some PDE applications, the matrices have little underlying structure that can be exploited (and any structure tends to vary from problem to problem), so we are forced to consider

a general iterative strategy.

This Chapter is arranged as follows. In §1.2, we will briefly describe the SPEEDUP package, including some of its applications. §1.3 will detail how SPEEDUP models are written, and how such a model is converted into a system of DAE's, and ultimately a system of nonlinear equations. The mathematical methods used to solve these nonlinear systems are discussed in §1.4. The need for real-time solution methods is introduced in §1.5, along with some measures designed to help achieve this. Finally in §1.6 the content of the thesis is presented.

1.2 SPEEDUP: A process modelling tool

The SPEEDUP software package is a modelling tool designed to be used in a wide range of industrial applications. It is used to simulate the operation of chemical plants and other process engineering applications, including petrochemicals, petroleum refining, gas processing, nuclear, pharmaceuticals, minerals, power, and food processing. The practical uses of SPEEDUP are wide and varied within the process engineering industry. Simulations can be used to improve the safety of plant operations by examining critical areas in SPEEDUP and determining what actions to take in the case of an emergency. Models can also increase the efficiency of a plant by performing dynamic optimisation of the simulation. This can result in lower emissions and waste produce. SPEEDUP can also be used in the design of new apparatus, by performing simulations before changes are made to a plant, to see the most effective way to achieve the goals of the new equipment.

1.3 Speedup models

1.3.1 Differential and Algebraic equations

SPEEDUP simulates the operation of whole chemical plants or just parts of them, by the mathematical modelling of a series of *units* linked together in a *flowsheet.* These

units usually represent the key pieces of apparatus in the plant, which can include:

- Distillation columns
- Continuously stirred tank reactors
- Tubular reactors
- Pumps
- Valves
- Pipe networks
- Adsorption beds
- Product feeds and outputs
- Controllers and sensors.

Each of the unit types listed above is modelled by a series of differential and algebraic equations (DAE's). These equations arise from the underlying physical properties of the model, and are derived using principles such as mass balance, energy balance, kinetics and transport. For a more detailed discussion of the application of DAE's in chemical engineering, see Byrne and Ponzi [12]. Although SPEEDUP users are able to write models in terms of DAE's, SPEEDUP is provided with a library of standard models, so it is possible to create a flowsheet from these without detailed knowledge of any of the underlying equations.

The number of equations associated with a model tends to obey the following physically reasonable rule: The simpler the behaviour of the unit, the fewer variables are associated with it, and the fewer equations are needed to create an accurate model. The units will typically only be connected to other units in a few places: For example, a reactor may have two feeds and one output. These flows may have two variables associated with them, say flow rate and concentration, giving a total of six 'connections' to other units, whereas the total number of equations needed to describe the reactor model may be much larger than this. Units may also have 'recycles'. These are connections going in

the opposite direction to the flow. This may also be thought of as feedback. Recycles potentially have a greater impact on the solution methods, which will be discussed later.

Each variable has associated with it upper and lower bounds on its value. These correspond to physical constraints - for example, a Kelvin temperature must have a lower bound of zero. Thus a flowsheet is translated by SPEEDUP into a system of DAE's:

$$\begin{aligned} f(y, \dot{y}, z, t) &= 0, \\ g(y, z, t) &= 0, \end{aligned} \tag{1.2}$$

where $\dot{y}$ represents $\frac{\partial y}{\partial t}$, and

$$\begin{aligned} y, \dot{y}, &\in \mathbb{R}^N,\ z \in \mathbb{R}^M, \\ f &: \mathbb{R}^{2N+M+1} \rightarrow \mathbb{R}^N, \\ g &: \mathbb{R}^{N+M+1} \rightarrow \mathbb{R}^M, \end{aligned} \tag{1.3}$$

subject to the bounds

$$L^y_{(i)} \leq y_{(i)} \leq U^y_{(i)}, \quad i = 1, \ldots N, \tag{1.4}$$

$$L^z_{(j)} \leq z_{(j)} \leq U^z_{(j)}, \quad j = 1, \ldots M. \tag{1.5}$$

Here, y and z denote the differential, or state, variables, with time represented by t. The equations also feature parameters that have user-defined values. These can be fixed at a constant value for the whole integration, or vary according to time. These are called *SET variables*, and are used to allow users to model the behaviour of many different dynamic phenomena. Examples range from the operation of a valve to the composition of a chemical feed into a reactor. Changes in these SET variables will affect the rest of the flowsheet, and such changes are often called 'disturbances'. Disturbances that cause a large change in the flowsheet are responsible for poor real-time performance, and this is discussed in §1.5. The *index* of a system of DAE's is defined as the number of times that all or part of (1.2) must be differentiated with respect to t in order to determine $\dot{y}$ as a continuous function of y and t. SPEEDUP is designed to solve index 1 systems of DAE's only.

1.3.2 Time integration of DAE's

To model how the flowsheet behaves over time, it is necessary to integrate the system of DAE's (1.2) with respect to t. This requires suitable initial conditions, which can either be completely prescribed by the author of the flowsheet, or calculated by performing an initialisation procedure. This requires that at least N variables (either state or algebraic) are prescribed, and solves system (1.2) with these prescribed variables and $t = 0$ to determine the unknown variables.

A detailed discussion of numerical methods for DAE's is presented by März [46], but for our purposes it is sufficient to understand the simplest backward difference scheme proposed by Gear [30] in 1971. This scheme is called the implicit Euler method, and would approximate $\dot{y}_i$, the derivative of y with respect to t at the ith integration step t_i, by

$$\dot{y}_i = \frac{y_i - y_{i-1}}{h}, \tag{1.6}$$

where $h = t_i - t_{i-1}$ is the integration step and y_i and y_{i-1} are the values of y at t_i and t_{i-1} respectively. If this difference approximation is applied to the DAE's (1.2), it yields a system of nonlinear equations in y_i and z_i:

$$\begin{aligned} f(y_i, \tfrac{y_i - y_{i-1}}{h}, z_i, t_i) &= 0, \\ g(y_i, z_i, t_i) &= 0, \end{aligned} \tag{1.7}$$

where y_{i-1} and t_i are known. This system must be solved at each timepoint t_i, subject to the bounds on y and z (1.4). The sequence of linear systems (1.1) arises at this point, and the efficient solution of (1.1) is obviously an important factor in the overall performance of the simulation. In fact for real-world SPEEDUP flowsheets, it accounts for up to 60% of the solution time. This integration scheme is used by Aspentech Consultants for real-time applications (§1.5), as it has order $O(h^2)$ local error, global error of order $O(h)$ and is absolutely stable, and as it is a fixed-step method, presents an even CPU load for the whole integration. Convergence results for more sophisticated backward difference formulas applied to DAE's are presented by Lötstedt and Petzold in [43] and they discuss the practical implications of such methods in [44]. Generally speaking, a k-step fixed-stepsize method has global error of order $O(h^k)$.

1.4 Solving systems of nonlinear equations

1.4.1 Newton's method

By far the most popular method for solving systems of nonlinear equations is the Newton method (see, for example, Dennis and Schnabel [18]). This method is well understood, with its local quadratic rate of convergence and its sensitivity to the initial guess. We now define the *Jacobian matrix* of a nonlinear system of equations $F(x)$:

Definition 1.1 *The Jacobian matrix at a point x of a continuous function $F : \mathbb{R}^n \to \mathbb{R}^n$ that is continuously differentiable at that point is defined as*

$$J(x)_{(i,j)} = F'(x)_{(i,j)} = \frac{\partial f_{(i)}}{\partial x_{(j)}} \in \mathbb{R}^{n \times n}, \tag{1.8}$$

where $f_{(i)}$ is the ith nonlinear equation of the system F and $x_{(j)}$ is the jth component of the point x.

It is straightforward to derive the Jacobian for the system of equations (1.7). If we consider the equation (1.8) in terms of the two functions f and g and the variables y, $\dot{y}$ and z, then by application of the chain rule, the Jacobian for the DAE system (1.2) can be written (neglecting t) as

$$J(y, \dot{y}, z) = \begin{pmatrix} \frac{\partial f(y,\dot{y},z)}{\partial y} + \frac{\partial \dot{y}}{\partial y}\frac{\partial f(y,\dot{y},z)}{\partial \dot{y}} & \frac{\partial f(y,\dot{y},z)}{\partial z} \\ \frac{\partial g(y,z)}{\partial y} & \frac{\partial g(y,z)}{\partial z} \end{pmatrix}. \tag{1.9}$$

When we apply the backward difference approximation for $\dot{y}$ (1.6) to this, the Jacobian can be written as

$$J(y, z) = \begin{pmatrix} \frac{\partial f(y,z)}{\partial y} + \frac{1}{h}\frac{\partial f(y,z)}{\partial \dot{y}} & \frac{\partial f(y,z)}{\partial z} \\ \frac{\partial g(y,z)}{\partial y} & \frac{\partial g(y,z)}{\partial z} \end{pmatrix}, \tag{1.10}$$

and it is this Jacobian that is used in the solution of the nonlinear equations (1.7). The derivatives can be prescribed analytically by the author of the model, or calculated numerically by SPEEDUP. A more detailed discussion of the derivation of Jacobians when methods of higher order than the approximation (1.6) are used is contained in [42]. The resulting Jacobian will be sparse, since units are only connected to other units by

a few variables. This is important, since the numerical solution of sparse linear systems requires special treatment, discussed later.

For notational convenience, we will simplify the expression of the systems of nonlinear equations we are solving by combining y and z in a single column vector

$$x = \begin{pmatrix} y \\ z \end{pmatrix} \in \mathbb{R}^n, \; n = N + M.$$

We can now simplify (1.7) to

$$\begin{aligned} F(x) &= 0, \\ F &: \mathbb{R}^n \to \mathbb{R}^n, \end{aligned} \tag{1.11}$$

with a corresponding Jacobian $J(x)$. Newton's method applied to our nonlinear system of equations can then be expressed as: Given $x^{(0)}$,

$$J(x^{(i)})\delta x = -F(x^{(i)}), \quad i = 0, 1, \ldots \tag{1.12}$$

and

$$x^{(i+1)} = x^{(i)} + \delta x. \tag{1.13}$$

A Newton iteration is usually terminated by considering the values of $\|F(x^{(i)})\|$ and/or $\|\delta x\|$. In particular, SPEEDUP will accept a Newton iterate $x^{(i)}$ when either

$$\|F(x^{(i)})\| < \texttt{ftol}$$

and/or

$$\|\delta x^{(i)}\| < \texttt{dxtol} + \texttt{reltol}\,\|x^{(i)}\|$$

hold, where `ftol`, `dxtol` and `reltol` are user specified tolerances. For full details of the termination criteria, the reader is refered to the SPEEDUP User Manual [2, pages 4.80–4.81].

SPEEDUP currently uses a direct method based on a sparse-LU factorisation to solve the linear Jacobian equations (1.12). Since the structure of the Jacobians will not change much from one Newton iteration to the next, and from one timestep to the next, such factorisation methods employ a strategy that allows factorisations of subsequent

matrices to be performed relatively cheaply, provided the new matrix is numerically suitable for the old factorisation. This is convenient, since the initial factorisation process can be quite slow; for a matrix of order n, a simple LU factorisation requires $O(n^3)$ operations, with no attention paid to the preservation of sparsity. However, should a matrix not be suited to the existing factorisation, an expensive refactorisation is required. The issues of sparse LU factorisation are discussed in greater detail in Chapter 4.

1.4.2 Block decomposition

In order to reduce the amount of time spent calculating the LU factors of the Jacobian matrices, a block decomposition is performed on the Jacobian. Since the time taken to factorise a matrix increases with the cube of its dimension, faster solution times would be possible if the Jacobian could be broken down into smaller blocks. One way that this can be achieved by the application of two routines, the Harwell MC21 routine [20] to obtain a diagonal free from zeros, followed by the Tarjan algorithm [64], which reduces a matrix to block lower-triangular form. Such a decomposition is, in general, non-unique. The portion of the matrix below these blocks is generally very sparse.

The application of this decomposition results in many smaller blocks that can be solved sequentially, thus reducing solution time. Often these diagonal blocks correspond to individual units, but units may be broken down into several blocks, and often several units are grouped together in a single block. When recycles are present in a model, this usually results in large blocks in the resulting decomposition. In general, if a connection appears below the diagonal of a Jacobian, then a corresponding recycle will appear above the diagonal, resulting in a group of variables that cannot be broken down further. One example of this is a distillation column. This is comprised of a series of trays, with connections to the trays situated above and below in the column. Although each tray is a separate unit, columns generally appear as a single block in a decomposition, as each tray has both a connection and a recycle. Examples of such blocks are shown in Chapter 3 (see, for example, Figure 3-1).

One potential problem with the use of such a decomposition is that of error propagation

- small errors introduced in the solution of the first block may be compounded by the solution of subsequent blocks. However, since the area below the block diagonal is sparse (corresponding to the small amount of connections between units compared to the overall number of variables per unit), then the contribution to blocks solved later in the system by inaccurate values will be negligible, and this is observed in practice.

It would be expensive to perform the decomposition process at each linear solve. Instead, SPEEDUP takes the approach one step further and applies the decomposition algorithms at the nonlinear equation level. This is achieved by forming an 'occurrence matrix', a symbolic representation of the nonlinear system, and forms a block decomposition of this. This results in a series of smaller nonlinear systems that can be solved sequentially. Indeed, many of these small nonlinear systems end up being one dimensional linear equations that can be solved trivially. However, large nonlinear blocks still occur, and it is these that require the largest proportion of the total time taken to complete the integration. For example, in the BTX model presented in Chapter 3, the flowsheet has a total of 1087 variables, and after decomposition, the largest nonlinear block is of size 927, still a large proportion of the total system size.

Strategies exist to allow the decomposition to produce smaller blocks - for example, careful structuring of the flowsheet can drastically improve the decomposition. The largest example presented in Chapter 4 has around 60,000 variables, and can be decomposed into one significant block of size 25,000 and many more smaller blocks.

1.5 Real-Time solution

1.5.1 Application: Operator training

Aspentech Consultants (ATC) is a division of Aspentech that offers consultancy services to the process engineering industry. These services include real-time modelling, and the provision of operator training systems. This requires real-time dynamic models of the key features of the plant that the operator is being trained for. The development of such systems allows experienced users to train new operators to manage chemical plants

without risking damage to expensive hardware, and potential environmental disaster, by simulating the operation of their plant with the dynamic model. One crucial aspect of this type of use is that SPEEDUP needs to be able to run in real-time, i.e. the time taken for the simulation to be completed should be the same as, or less than, the elapsed period of time in the model. A plot of the CPU time versus real-time for an integration that does not run in real-time is shown in Figure 1-1. Whilst the use of parallel supercomputers could enable real-time solution, often the software is required to run on single workstations. For example, ATC has a contract with Mitsubishi Kasai Corporation for the development of a large scale real-time dynamic simulator for a key petrochemicals plant. This requires the modelling of over 600 equipment items with potentially hundreds of thousands of variables to run in real-time on an IBM RS/6000.

As we saw in the previous section, the most expensive phase of the solution process is the solution of the linear systems of equations (1.12) arising from Newton's method applied to (1.7). As long as the new Jacobians do not differ too much from the previously factored one, the new LU factors can be calculated quite cheaply. However, when using Newton's method, the Jacobians change at each Newton iteration, and so many refactorisations may be required. If a flowsheet has large disturbances (§1.3.1) present during the integration period, then this increases the number of refactorisations required even further. On problems with large block sizes, these refactorisations can have a large impact on a simulation's ability to run within real-time. Typically, if several refactorisations of a large block are required on one timestep, then this can result in the solution time being many times that of real-time for that timestep.

1.5.2 Approximate Newton method

To try and avoid large numbers of factorisations during the solution process, SPEEDUP has an alternative nonlinear solver option, called FASTNEWTON. This uses a variation of the Newton method. In this, the standard Newton's method ((1.12), (1.13)) is replaced by the following: The correction step δx is calculated from

$$J(x^{(0)})\delta x = -F(x^{(i)}), \tag{1.14}$$

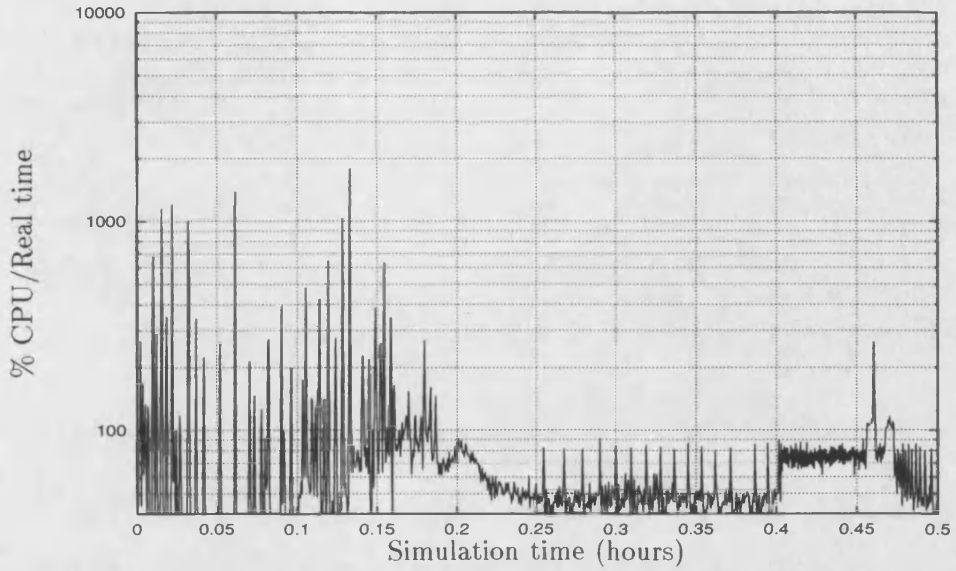

Figure 1-1: A typical CPU/realtime plot for a SPEEDUP integration.

and

$$x^{(i+1)} = x^{(i)} + \delta x.$$

Kelley [38] calls this the Chord Method, and shows that this method possesses linear convergence properties when the initial guess $x^{(0)}$ is close enough to the exact solution of (1.11). This approach is motivated by the fact that the same matrix $J(x^{(0)})$ is used for each Newton iteration. This means that once a factorisation of $J(x^{(0)})$ has been calculated, then the subsequent Newton iterates can be obtained cheaply without the need for additional factorisations. This comes at the cost of additional Newton steps, since the linear convergence of the approximate method is not as fast as the quadratic convergence of the standard Newton method. SPEEDUP will retain the Jacobian from the initial timestep for as long as possible,

Unfortunately, the approximate method may not converge rapidly enough in some situations (SPEEDUP imposes an upper limit on the number of iterations allowed per Newton step, usually 10). When this occurs, the solution method resorts to the standard Newton method for the remainder of the current nonlinear block. This means that many refactorisations may still be required. If the standard Newton method was needed for the current timestep, then SPEEDUP reverts to approximate Newton for the next timestep, using the last Jacobian to be factorised. The disturbances in the flowsheet

show up clearly in Figure 1-1 as the large 'spikes' in the CPU/real-time ratio, with CPU time often in excess of 1000% of real-time.

1.5.3 Appeal to iterative methods

Recently, modern iterative methods for the solution of systems of linear equations

$$Ax = b$$

have been shown to be highly effective in the solution of sparse systems arising from many areas of applied maths. The majority of these methods belong to the family of Krylov subspace methods, with perhaps the most famous being the Conjugate Gradient method proposed by Hestenes and Stiefel in 1952 [35]. They rely on matrix-vector products, and so are suited to problems where A is sparse, since the products will be cheap. These methods generally compete better with direct methods when the dimension n of A is large, as they are not subject to the $O(n^3)$ operations count like direct methods. Instead, convergence is governed by the distribution of the spectrum of A. Many such methods also possess some form of optimality property, providing some guarantee of robustness.

The motivation for this project was to see if such methods could prove effective in reducing the occurrence and/or size of the spiking effect caused by the refactorisations needed by the sparse-LU method currently in use, thus producing reduced solution times.

Whilst such a method may be more expensive than a direct method when the integration is proceeding without refactorisations, it may be able to perform better than the direct method when disturbances occur in the flowsheet by avoiding the need for the additional factorisations. The project called for a general iterative method, coupled with a suitable preconditioner, since due to the design of the flowsheet and the use of block-decomposition, there would be no structure for such a method to take advantage of. This thesis contains an account of the work carried out on this project.

1.6 Overview of Thesis and results

We now present an overview of the rest of the thesis.

In Chapter 2, Krylov subspace iterative methods are introduced, focusing on the GMRES algorithm [61]. Existing results regarding the convergence of the algorithm in the residual norm, and novel results regarding convergence in the error norm are presented. The effect on the convergence of groups of small and large outlying eigenvalues is discussed, and improved bounds are presented for these cases. Finally, some of the more common problems with the GMRES algorithm are introduced, with strategies for detecting and countering them.

In Chapter 3, preconditioners are introduced. The need for preconditioning for real-world problems is discussed, along with the requirements that must be fulfilled by a preconditioner. Four test examples from SPEEDUP problems are presented. The poor performance of ILU preconditioned GMRES is discussed and analysed, with the aid of pseudospectral analysis. An alternative preconditioner based on a complete LU factorisation is developed, taking advantage of the fact that we are solving a sequence of linear systems (1.1). The SPEEDUP test matrices and pseudospectral methods are used again to demonstrate the numerically superior properties of this approach.

In Chapter 4, the preconditioner developed in the previous chapter is used to derive the FLUGMR algorithm, using GMRES as a black-box solver in SPEEDUP integration runs. The requirements of such a solver are discussed, and the method of calculating the preconditioner is presented. Results from three SPEEDUP flowsheets are presented, demonstrating the robustness of the solver. The results are used to suggest optimal strategies for the solver. The results are compared to those of the direct solver, and profile runs are used to understand why FLUGMR is not as fast as the best direct method.

Inexact Newton methods are discussed in Chapter 5. The motivation for this approach is the use of iterative methods for the solution of (1.12) and observed disagreement between the values of the nonlinear function F and its local linear model, especially during the initial Newton iterations. The inexact Newton algorithm is presented, and

the role of the forcing term is introduced. Several different choices for the forcing term are used, and convergence properties of each type are given. Results from SPEEDUP integrations are presented, and the performances of the various choices are discussed.

Implementations of iterative methods other than GMRES are tested in Chapter 6. CGN, BCG-type methods and QMR variants are all tried, and results presented. The poor performance of these methods is described, along with their potential causes.

Finally, conclusions and directions for future work are discussed in Chapter 7.

Chapter 2

Iterative methods for non-symmetric linear systems

2.1 Introduction

In this chapter, we present a brief summary of three popular types of Krylov subspace methods, and go on to examine one specific method, the GMRES algorithm [61], that will be the main focus of this thesis. We present existing results regarding convergence of the algorithm, and novel results concerning convergence in the error. We examine one of the numerical problems with the standard GMRES implementation, and mention some strategies to combat this.

2.2 Krylov subspace methods

In recent years, many new iterative methods for solving non-symmetric linear systems of the form

$$Ax = b, \tag{2.1}$$

$A \in \mathbb{R}^{n \times n}$, $x, b \in \mathbb{R}^n$ have been proposed (see Freund *et al.* [28] for a historical survey of these developments). Many of these methods have a common property: They all use

the Krylov subspace of A, which for $k \in \mathbb{N}$ and a given vector v is defined by

$$\mathcal{K}_k(A, v) = \text{span}\{v, Av, A^2v, \ldots, A^{k-1}v\}, \tag{2.2}$$

and produce iterates using elements of this subspace,

$$x_k = x_0 + z,\ z \in \mathcal{K}_k(A, v), \tag{2.3}$$

where x_0 is the initial estimate of the solution. The initial subspace vector is taken as

$$v = r_0 = b - Ax_0,$$

the initial residual in all of the methods considered here (in fact, most of the algorithms begin with $v = \frac{r_0}{\|r_0\|}$). In general terms, these iterative methods can be categorised into three types:

a) Methods based on the normal equations,

b) Methods employing orthogonalisation techniques, and

c) Methods employing biorthogonalisation techniques.

Examples of each of these types are given below (for a more complete list, see for example Nachtigal, Reddy and Trefethen [49]):

a) CGN - Conjugate gradients applied to the normal equations

$$A^TAx = A^Tb. \tag{2.4}$$

This method was originally proposed in the original CG paper by Hestenes and Stiefel in 1952 [35].

b) GMRES - Generalised minimal residuals [61]. This method, proposed by Saad and Schultz, uses the Arnoldi process [1] to produce an l_2 orthonormal basis for $\mathcal{K}_k$, and its iterates possess an optimality property for the norm of the residual.

c) BCG - Biconjugate gradients. This method produces non-optimal approximate solutions in the same subspace as GMRES, but the iterates are cheaper to produce.

One common feature of all these methods is their reliance on matrix-vector products with A. Note that A is not needed explicitly, merely its action on a vector, so such methods are sometimes called *matrix free*. Note also that these methods are suited to problems where matrix-vector products are available cheaply, such as sparse-matrix problems, where direct solution methods may be prohibitively expensive.

Each class of method comes with its own set of advantages and disadvantages. This means that there is no one solver that is best for all types of problem. Nachtigal, Reddy and Trefethen [49] construct a series of examples on which each of methods a), b) and c) significantly outperforms the other two. One feature they have in common, however, is that for normal or slightly non-normal matrices, convergence is dependent on the distribution of the eigenvalues of the matrix to which the method is applied; for CGN this means that convergence is governed by the *singular values* of A. For highly non-normal matrices, this is not neccesarily true. Nachtigal, Reddy and Trefethen [49] state that for these cases, the eigenvalues of the matrices alone may not provide sufficient information to accurately predict convergence. A more extreme view is presented by Greenbaum, Pták and Strakoš [32], who state that eigenvalues are not at all relevant to the rate of convergence for GMRES when applied to highly non-normal matrices. Whilst this may be the case for specifically constructed examples, the numerical results presented in Chapter 3 indicate that, for SPEEDUP problems, eigenvalues do provide useful information about the convergence of GMRES.

As indicated in the previous paragraph, the method we choose to use as the iterative linear solver in SPEEDUP is the GMRES method. One of the major requirements of an iterative solver to be implemented in SPEEDUP is robustness: A breakdown of the iterative solver during a time integration would leave the integration incomplete. GMRES cannot break down and converges monotonically in the residual norm. These two properties made GMRES the sensible choice for an iterative solver in SPEEDUP. Due to the coordinate storage method used for the SPEEDUP Jacobians, the action of A^T on a vector is freely available, however, results presented in Chapter 6 show that solvers

of type a) are unsuitable for SPEEDUP Jacobian problems. The class of biorthogonalisation methods (type c)) are susceptible to breakdown or erratic convergence behaviour, and so were not such attractive alternatives for the applications considered here, where robustness is important. Nonetheless, these too are considered in Chapter 6.

In the remainder of the chapter, we will present the GMRES method in greater detail, examining the algorithm and aspects of its convergence behaviour in both the residual and the error, and show how the distribution of the spectrum of the matrix A can drastically effect the rate of convergence of the algorithm.

2.3 The GMRES Algorithm

2.3.1 Introduction

The Generalised Minimal RESidual algorithm, or GMRES, was proposed in 1986 by Saad and Schultz [61] as a method of solving non-symmetric linear systems

$$Ax = b, \tag{2.5}$$

with $A \in \mathbb{R}^{n\times n}, x, b \in \mathbb{R}^n$. GMRES is an iterative method, producing a sequence of iterates $\{x_k\}$ that minimise the l_2 norm of the residual

$$r_k = b - Ax_k$$

over the Krylov subspace (2.2) of A, $\mathcal{K}_k(A, r_0)$. The minimal residual property can be expressed as (see, for example, Kelley [38])

$$\|b - Ax_k\|_2 = \min_{x\in x_0+\mathcal{K}_k} \|b - Ax\|_2. \tag{2.6}$$

2.3.2 Arnoldi's method and the FOM

It is helpful to describe briefly the background of the GMRES algorithm, and to do this we need to know about Arnoldi's method. Arnoldi's method [1] is a Galerkin method for approximating the eigenvalues λ of a matrix A by the eigenvalues $\hat{\lambda}$ of an upper Hessenberg matrix H_k. The matrix H_k arises from the construction of an l_2-orthogonal basis for the Krylov subspace of A by the Gramm-Schmidt process (see for example [59])

Algorithm 2.1 *Arnoldi's method*

1. **Start:** Choose v_1 such that $\|v_1\|_2 = 1$.

2. **Iterate:** For $j = 1, 2, \ldots k$ do:
   $h_{i,j} = (Av_j, v_i), i = 1, 2, \ldots j,$
   $\hat{v}_{j+1} = Av_j - \sum_{i=1}^{j} h_{i,j} v_i,$
   $h_{j+1,j} = \|\hat{v}_{j+1}\|_2,$ and
   $v_{j+1} = \hat{v}_{j+1}/h_{j+1,j}.$

So H_k is simply the upper Hessenberg matrix with $(H_k)_{i,j} = h_{i,j}$ from step 2. In practice, the Gramm-Schmidt orthogonalisation in step 2 is usually replaced by the mathematically equivalent but numerically superior modified Gramm-Schmidt process [31].

If we denote by V_k the $n \times k$ matrix whose columns are the constructed basis for $\mathcal{K}_k$,

$$V_k = [v_1, v_2, \ldots, v_k], \tag{2.7}$$

then A, V_k and H_k have the relation

$$H_k \equiv V_k^T A V_k.$$

In 1981, Saad proposed a method for solving linear systems (2.1) based on the Arnoldi method. The Full Orthogonalisation Method (FOM) [57] uses the Arnoldi process with $v_1 = r_0/\|r_0\|_2$, and produces iterates by imposing the Galerkin condition that the residual be l_2-orthogonal to the subspace $\mathcal{K}_k(A, r_0)$. This results in the solution of a $k \times k$ linear system

$$H_k y = \beta e_1, \tag{2.8}$$

where $\beta = \|r_0\|_2$. Since H_k is upper Hessenberg, (2.8) can be solved easily using Givens rotations [31]. The iterates x_k are given by

$$x_k = x_0 + V_k y.$$

Although the FOM algorithm has a finite termination property, its iterates do not have any kind of optimality property, and the work and storage requirements grow with k. Saad suggests restarting the method or performing only incomplete factorisation (IOM) [57, 58] to avoid the iteration becoming too expensive. The lack of an optimality property for the FOM led to the development of a new iterative method, namely GMRES.

2.3.3 GMRES

In 1986, Saad and Schultz proposed the GMRES algorithm [61]. This has the attraction that it possesses an optimality property for the norm of the residual, whilst its iterates are still obtained from the same underlying Arnoldi process as the FOM. If we consider the kth step of the Arnoldi algorithm, then we actually obtain V_{k+1} and a $(k+1) \times k$ matrix $\bar{H}_k$ has H_k as an upper $k \times k$ block and an extra row whose only non-zero entry is $h_{k+1,k}$. These matrices satisfy the relation:

$$AV_k = V_{k+1}\bar{H}_k. \tag{2.9}$$

Since x_k can be expressed as $x_0 + z, z \in \mathcal{K}_k$, the optimality property (2.6) becomes

$$\|b - Ax_k\|_2 = \min_{z \in \mathcal{K}_k} \|b - A(x_0 + z)\|_2 = \min_{z \in \mathcal{K}_k} \|r_0 - Az\|_2. \tag{2.10}$$

Since $z \in \mathcal{K}_k$, we can express it in the form

$$z = V_k y. \tag{2.11}$$

Substituting this into (2.10) and taking our initial vector $v_1 = r_0/\|r_0\|_2$, $\beta = \|r_0\|_2$, gives the norm we are minimising as a function of y:

$$J(y) = \|\beta v_1 - AV_k y\|_2. \tag{2.12}$$

Using (2.9), this can be rewritten as

$$J(y) = \|V_{k+1}[\beta e_1 - \bar{H}_k y]\|_2 \tag{2.13}$$

where e_1 is the first column of the $(k+1) \times (k+1)$ identity matrix. Now V_{k+1} is l_2-orthonormal so we can simplify (2.13) to

$$J(y) = \|\beta e_1 - \bar{H}_k y\|_2. \tag{2.14}$$

As with the FOM, the solution of (2.14) can be obtained cheaply by maintaining and updating a least-squares factorisation of $\bar{H}_k$ on each GMRES iteration. This is easily achieved using Givens rotations, since $\bar{H}_k$ is upper Hessenberg, and merely entails the application of rotations calculated on previous iterations to the new column generated at the current iteration before applying the new rotation. An storage-efficient implementation of this approach is presented in [63].

Thus the minimisation problem has been reduced from an $n \times k$ least-squares problem (2.10) to a $(k+1) \times k$ one (2.14). This gives us the following algorithm:

Algorithm 2.2 *Generalised Minimal Residual method (GMRES)*

1. **Start:** Choose x_0 and compute $r_0 = b - Ax_0$. Set $v_1 = r_0/\|r_0\|_2$
2. **Iterate:** For $j = 1, 2, \ldots k, \ldots$ until satisfied do:
   $h_{i,j} = (Av_j, v_i)$, $i = 1, \ldots j$,
   $\hat{v}_{j+1} = Av_j - \sum_{i=1}^{j} h_{i,j} v_i$,
   $h_{j+1,j} = \|\hat{v}_{j+1}\|_2$, and
   $v_{j+1} = \hat{v}_{j+1}/h_{j+1,j}$.
3. **Form the approximate solution:** $x_k = x_0 + V_k y_k$, y_k minimises (2.14)

As was the case with FOM, the amount of work and storage required on each iteration grows with k. As the number of multiplications required on each step grows like $\frac{1}{2}k^2 n$, this results in prohibitively slow performance for large values of k. In the original paper [61], Saad and Schultz suggest using the procedure in a restarted fashion, restricting the maximum number of inner iterations to some value m. They denote this restarted version by GMRES(m), and it is described as follows:

Algorithm 2.3 *GMRES(m)*

1. **Start:** Choose x_0 and compute $r_0 = b - Ax_0$. Set $v_1 = r_0/\|r_0\|_2$
2. **Iterate:** For $j = 1, 2, \ldots m$ do:
   $h_{i,j} = (Av_j, v_i)$, $i = 1, \ldots j$,
   $\hat{v}_{j+1} = Av_j - \sum_{i=1}^{j} h_{i,j} v_i$,
   $h_{j+1,j} = \|\hat{v}_{j+1}\|_2$, and
   $v_{j+1} = \hat{v}_{j+1}/h_{j+1,j}$.
3. **Form the approximate solution:** $x_m = x_0 + V_m y_m$, y_m minimises (2.14)
4. **Restart:** Compute $r_m = b - Ax_m$; if satisfied then stop
   else set $x_0 \leftarrow x_m$, $v_1 = r_m/\|r_m\|_2$ and goto 2.

This restarted algorithm has no finite convergence property (see Theorem 2.1), but this is only of theoretical interest since we are aiming to produce accurate approximate solutions x_k in $k \ll n$ iterations. It has been observed that the algorithm can be very sensitive to the choice of the subspace length m. Huang and van der Vorst [36] have several examples whereby a slight increase in m produces a large improvement in the performance of the algorithm. They also suggest that although GMRES demonstrates superlinear convergence behaviour - the speed of convergence increases as the iteration proceeds (see [68]) - GMRES(m) does not exhibit this behaviour.

2.3.4 Residual norm estimates for free

One useful feature of the GMRES algorithm is that it is possible to monitor its performance without explicitly forming the approximate solution at each step. If we recall the system we are required to solve at each GMRES iteration (2.14), then this is usually performed by a QR factorisation [31] of $\bar{H}_k$,

$$QR = \bar{H}_k,$$

where Q is orthonormal and R is upper rectangular. We can then re-write (2.14) as

$$\begin{aligned} J(y) = \|\beta e_1 - \bar{H}_k y\|_2 &= \|\beta e_1 - QRy\|_2 \\ &= \|Q[Q^T\beta e_1 - Ry]\|_2 \\ &= \|Q^T\beta e_1 - Ry\|_2. \end{aligned} \tag{2.15}$$

We are seeking y_k that minimises the function $J(y)$, which is found by solving the upper triangular system formed by removing the row of zeros from the bottom of R and the last component of the transformed right-hand side, $Q^T\beta e_1$. We can now see that the value of $J(y_k)$ is equal to the absolute value of the last component of $Q^T\beta e_1$ by construction of y_k. So the residual norm is available 'for free' once the least-squares problem (2.13) has been solved. In exact arithmetic, this value is exactly the residual norm, but in finite precision this is not necessarily so. If the basis vectors $\{v_i,\ i = 1, \ldots, k+1\}$ are not l_2-orthonormal, then the matrix V_{k+1} (2.7) will not be either. In this case, we have

$$\|V_{k+1}[\beta e_1 - \bar{H}_k y]\|_2 \neq \|\beta e_1 - \bar{H}_k y\|_2,$$

and so y_k that minimises (2.14) will not minimise (2.12), and the estimates of the residual norms given by the algorithm will not be equal to the true residual norms. Loss of orthogonality in the basis vectors and strategies to counter this are discussed in §2.7

We now present some standard results due to Kelley [38] concerning the convergence behaviour of Algorithm 2.2 with respect to the norm of the residual.

2.4 Convergence Behaviour I: Residuals

2.4.1 Residual polynomials

A key issue with any iterative method is how fast the approximate solution converges to the exact solution. Due to the nature of the Krylov subspace (2.2), it is possible to

expand the kth iterate $x_k \in x_0 + \mathcal{K}_k(A, r_0)$ in terms of a polynomial in A:

$$x_k = x_0 + \sum_{i=0}^{k-1} \gamma_i A^i r_0,$$

and similarly for the residual:

$$b - Ax_k = b - Ax_0 - \sum_{i=0}^{k-1} \gamma_i A^{i+1} r_0 = r_0 - \sum_{i=1}^{k} \gamma_{i-1} A_i r_0.$$

To continue the analysis, we define the set of *kth* degree residual polynomials;

Definition 2.1 *The set of kth degree residual polynomials is given by:*

$$\mathcal{P}_k = \{p |\ p \in \mathbb{P}_k,\ p(0) = 1\}$$

where $\mathbb{P}_k$ is the set of polynomials of degree not exceeding k.

Thus we can write

$$r_k = p(A) r_0,$$

where $p \in \mathcal{P}_k$ is a residual polynomial. We now have

Lemma 2.1 *Let A be nonsingular and $r_k = b - Ax_k$ be the residual from the kth GMRES iteration. Then for all $\bar{p}_k \in \mathcal{P}_k$*

$$\|r_k\|_2 \leq \|\bar{p}_k(A) r_0\|_2. \tag{2.16}$$

Proof. $r_k = p(A) r_0$ implies

$$\|r_k\|_2 = \min_{p \in \mathcal{P}_k} \|p(A) r_0\|_2,$$

so clearly

$$\|r_k\|_2 \leq \|\bar{p}_k(A) r_0\|_2$$

for all $\bar{p}_k \in \mathcal{P}_k$. □

Immediately from this we have the corollary

Corollary 2.1 *Under the same assumptions as Lemma 2.1,*

$$\frac{\|r_k\|_2}{\|r_0\|_2} \leq \|\bar{p}_k(A)\|_2. \tag{2.17}$$

2.4.2 Finite convergence of GMRES

We are now in a position to show that GMRES will converge (in exact arithmetic) to the exact solution in at most n steps (finite convergence property):

Theorem 2.1 *Let $A \in \mathbb{R}^{n\times n}$ be a nonsingular matrix. Then the GMRES approximate solution will converge to the exact solution in at most n iterations.*

Proof. If we consider the characteristic polynomial of A, $P(z) = \det(A - zI)$, then $P \in \mathbb{P}_n$ and $P(0) \neq 0$ since A is nonsingular, so we can define an nth-degree residual polynomial of the form

$$p_n(z) = P(z)/P(0).$$

We can now write

$$\frac{\|r_n\|_2}{\|r_0\|_2} \leq \|p_n(A)\|_2,$$

but by the Cayley-Hamilton theorem, we have that $P(A) = 0$, so $p_n(A) = 0$ and therefore $r_n = b - Ax_n = 0$ and x_n is the exact solution. □

Note that for the restarted version GMRES(m) (Algorithm 2.3), we cannot apply Theorem 2.1 as we restrict the length of the subspace to $m < n$ and so cannot form the polynomial $p_n(z)$.

2.4.3 Rates of Convergence

Whilst the finite convergence property gives us the reassurance that the algorithm will terminate eventually, it has already been noted that GMRES in its non-restarted form is prohibitively expensive. It is therefore desirable to obtain some estimate of how fast the algorithm is converging. If the matrix A is diagonalisable, i.e. if there is a nonsingular

(complex) matrix U such that

$$A = U\Lambda U^{-1}$$

where Λ is a (complex) diagonal matrix with the eigenvalues $\lambda \in \sigma(A)$ on the diagonal, then we can re-express polynomials in A as

$$p(A) = Up(\Lambda)U^{-1}.$$

From Corollary 2.1 we can now obtain

Theorem 2.2 *Assume A is a nonsingular diagonalisable matrix with $A = U\Lambda U^{-1}$. Let $r_k = b - Ax_k$ be the residual from the kth GMRES iteration. Then for all $\bar{p}_k \in \mathcal{P}_k$*

$$\frac{\|r_k\|_2}{\|r_0\|_2} \leq \kappa_2(U) \max_{\lambda \in \sigma(A)} |\bar{p}_k(\lambda)|. \tag{2.18}$$

Proof. The result follows from (2.17) and

$$\|\bar{p}_k(A)\|_2 \leq \|U\|_2 \|\bar{p}_k(\Lambda)\|_2 \|U^{-1}\|_2 \leq \kappa_2(U) \max_{\lambda \in \sigma(A)} |\bar{p}_k(\lambda)|.$$

□

Note that if A is normal then the matrix U is orthogonal, and $\kappa_2(U) = 1$.

With $\mathcal{P}_k$ as in Definition 2.1, we define

$$\epsilon^{(k)} \equiv \min_{p \in \mathcal{P}_k} \max_{\lambda \in \sigma(A)} |p(\lambda)|. \tag{2.19}$$

This allows us to state

Theorem 2.3 *Assume that A is a nonsingular and diagonalisable matrix of the form $A = U\Lambda U^{-1}$. Let r_k be the residual from the kth GMRES iteration. Then*

$$\frac{\|r_k\|_2}{\|r_0\|_2} \leq \kappa_2(U)\epsilon^{(k)}. \tag{2.20}$$

Proof. Since from (2.18) we have for all $\bar{p}_k \in \mathcal{P}_k$

$$\frac{\|r_k\|_2}{\|r_0\|_2} \leq \kappa_2(U) \max_{\lambda \in \sigma(A)} |\bar{p}_k(\lambda)|,$$

then clearly

$$\frac{\|r_k\|_2}{\|r_0\|_2} \leq \kappa_2(U) \min_{p \in \mathcal{P}_k} \max_{\lambda \in \sigma(A)} |p_k(\lambda)|.$$

□

This Theorem allows us to obtain a bound on the rate of reduction in the residual norm by considering the value of $\epsilon^{(k)}$. In general, the calculation of $\epsilon^{(k)}$ is difficult. Instead, we can obtain an upper bound, $\eta^{(k)}$ say, by considering a maximum over a complex domain D which contains $\sigma(A)$ and excludes 0. If $\sigma(A) \subset D$ then clearly

$$\max_{\lambda \in \sigma(A)} |p(\lambda)| \leq \max_{z \in D} |p(z)|, \tag{2.21}$$

and so

$$\epsilon^{(k)} \leq \eta^{(k)} = \min_{p \in \mathcal{P}_k} \max_{z \in D} |p(z)|. \tag{2.22}$$

If we consider specific domains D (and therefore specific distributions of $\sigma(A)$), we can determine $\eta^{(k)}$ exactly by means of scaled and shifted Chebyshev polynomials.

2.4.4 Chebyshev polynomials

We can define the Chebyshev polynomials as

Definition 2.2 *The kth degree Chebyshev polynomial in a real variable t is defined as*

$$T_k(t) = \begin{cases} \cos(k \cos^{-1} t) & \text{when } |t| \leq 1 \\ \cosh(k \cosh^{-1} t) & \text{when } |t| > 1. \end{cases} \tag{2.23}$$

The kth degree Chebyshev polynomial in a complex variable z is defined as

$$T_k(z) = \cosh(k \cosh^{-1} z). \tag{2.24}$$

In the following theorem, three domains D are considered:

(a) a line segment,

(b) a disc,

(c) the interior of an ellipse with real major axis.

Theorem 2.4 *Define*

$$\eta^{(k)} = \min_{p \in \mathcal{P}_k} \max_{z \in D} |p(z)|.$$

Then for the following domains D, $\eta^{(k)}$ is characterised by

(a) *When D is the real interval $\{t; |t - c| \leq a, t \neq 0\}$,*

$$\eta^{(k)} = \frac{1}{T_k\left(\frac{c}{a}\right)}.$$

(b) *When D is the disc $\{z; |z - c| \leq \rho, z \neq 0\}$*

$$\eta^{(k)} = \left(\frac{\rho}{c}\right)^k.$$

(c) *When D is bounded by the ellipse with centre c, focal distance e and semi-major axis a,*

$$\eta^{(k)} = \frac{T_k(a/e)}{T_k(c/e)},$$

where a, c, e and ρ are positive real numbers.

Proof. See Chatelin [14] Theorem 6.6.2. □

This Theorem allows us to obtain estimates for the rate of reduction of the residual norm for cases where we know either the exact distribution of the spectrum $\sigma(A)$ or a domain D which contains the distribution. Unfortunately, convergence of the residual is not always a good indication of how the iteration is proceeding. If we consider the relation

$$r_k = Ae_k, \tag{2.25}$$

where $e_k = x - x_k$, the error at the kth GMRES iteration, then we can see that if the error has a large component in the direction of an eigenvector corresponding to a small eigenvalue of A, then it would be possible for the norm $\|r_k\|_2$ to be small even when the norm $\|e_k\|_2$ is still large. Ideally, we would like to consider the convergence of the error, rather than the residual, as the GMRES iteration proceeds.

2.5 Convergence Behaviour II: Errors

2.5.1 The distance of x from the subspace

In this section we present theoretical error bounds for GMRES. We begin by considering the distance d_k of the exact solution x to the Krylov subspace generated by the GMRES algorithm, $\mathcal{K}_k$:

$$d_k = \|(I - \pi_k)x\|_2, \tag{2.26}$$

where π_k is the orthogonal projection matrix for $\mathcal{K}_k$. For the Arnoldi process [1] to solve the algebraic eigenvalue problem $Au = \lambda u$, Chatelin [14] and Saad [59] relate the distance $\|(I - \pi_k)u\|_2$, the distance of an exact eigenvector u from the Krylov subspace, to an expression similar to $\epsilon^{(k)}$ defined in (2.19).

If we assume that A is diagonalisable and nonsingular, we can relate the distance d_k to $\epsilon^{(k)}$ in an analogous manner:

Lemma 2.2 *Assume that A is non-singular and diagonalisable, and that the solution and right-hand side vectors x and b have expansions $x = \sum_{i=1}^{n} \alpha_i u_i$ and $b = \sum_{i=1}^{n} \beta_i u_i$ with respect to the eigenbasis $\{u_i\}_{i=1,\ldots n}, \|u_i\|_2 = 1$ of A. Assume without loss of generality that $\|b\|_2 = 1$ and take an initial guess $x_0 = 0$. Then the distance of the exact solution x from the Krylov subspace $\mathcal{K}_k$ generated by GMRES can be bounded by*

$$d_k \leq \xi \epsilon^{(k)} \tag{2.27}$$

where

$$\xi = \sum_{i=1}^{n} \frac{|\beta_i|}{|\lambda_i|}.$$

Proof. Since $x_0 = 0$, we have $r_0 = b$. Now $v_1 = \frac{r_0}{\|r_0\|_2} = b$ since $\|b\|_2 = 1$. Recalling that $Ax = b$, we can write

$$\begin{aligned} b = \sum_{i=1}^{n} \beta_i u_i &= A \sum_{i=1}^{n} \alpha_i u_i \\ &= \sum_{i=1}^{n} \alpha_i \lambda_i u_i \end{aligned}$$

which gives us

$$\beta_i = \alpha_i \lambda_i. \tag{2.28}$$

From the relation between $\mathcal{K}_k$ and $\mathbb{P}_{k-1}$, we have

$$\begin{aligned} \|(I - \pi_k)x\|_2 &= \min_{q \in \mathbb{P}_{k-1}} \|x - q(A)v_1\|_2 \\ &= \min_{q \in \mathbb{P}_{k-1}} \|\sum_{i=1}^{n} \alpha_i u_i - q(A) \sum_{i=1}^{n} \beta_i u_i\|_2 \\ &= \min_{q \in \mathbb{P}_{k-1}} \|\sum_{i=1}^{n} [\alpha_i - \beta_i q(\lambda_i)] u_i\|_2 \\ &= \min_{q \in \mathbb{P}_{k-1}} \|\sum_{i=1}^{n} \frac{\beta_i}{\lambda_i} [1 - \lambda_i q(\lambda_i)] u_i\|_2 \\ &\leq \min_{q \in \mathbb{P}_{k-1}} \sum_{i=1}^{n} \|\frac{\beta_i}{\lambda_i} [1 - \lambda_i q(\lambda_i)] u_i\|_2 \\ &\leq \min_{q \in \mathbb{P}_{k-1}} \sum_{i=1}^{n} |\frac{\beta_i}{\lambda_i} [1 - \lambda_i q(\lambda_i)]| \\ &\leq \min_{q \in \mathbb{P}_{k-1}} \max_{\lambda \in \sigma(A)} |1 - \lambda q(\lambda)| \sum_{i=1}^{n} \frac{|\beta_i|}{|\lambda_i|} \\ &= \min_{p \in \mathcal{P}_k} \max_{\lambda \in \sigma(A)} |p(\lambda)| \sum_{i=1}^{n} \frac{|\beta_i|}{|\lambda_i|}. \end{aligned}$$

□

We have insisted that $x_0 = 0$ in the above Lemma. This choice can be justified by an appeal to Huang and van der Vorst [36], who note that their experiments with both GMRES and GMRES(m) showed both algorithms to be quite insensitive to the initial

residual (and hence the initial guess) unless x_0 was selected such that r_0 is deficient in some eigenvector directions.

Since GMRES minimises the residual and not the error on $\mathcal{K}_m$, we cannot use the bound (2.27) to directly monitor the behaviour of the error since, in general, the GMRES iterate $x_k \neq \pi_k x$. Instead, we derive an error bound in terms of the distance d_k by exploiting the GMRES minimisation property:

Lemma 2.3 *The error at the kth step of the GMRES algorithm applied to the linear system $Ax = b$ with A nonsingular is bounded by*

$$\|e_k\|_2 \leq \kappa_2(A) d_k. \tag{2.29}$$

Proof. Consider the error at the kth step. We can write

$$e_k = x - x_k = A^{-1}A(x - x_k) = A^{-1}r_k, \tag{2.30}$$

and taking norms yields

$$\|e_k\|_2 \leq \|A^{-1}\|_2 \|r_k\|_2. \tag{2.31}$$

Now we know that $\|r_k\|_2$ from the GMRES algorithm satisfies

$$\begin{aligned} \|r_k\|_2 &= \min_{\bar{x} \in \mathcal{K}_k} \|b - A\bar{x}\|_2 \\ &= \min_{\bar{x} \in \mathcal{K}_k} \|A(x - \bar{x})\|_2 \\ &\leq \|A\|_2 \min_{\bar{x} \in \mathcal{K}_k} \|x - \bar{x}\|_2 \\ &= \|A\|_2 d_k. \end{aligned}$$

So now we have

$$\|e_k\|_2 \leq \|A\|_2 \|A^{-1}\|_2 d_k$$

which completes the proof. □

We are now in a position to derive an initial bound on the size of the error:

Theorem 2.5 *Assume that A is non-singular and diagonalisable, and that ξ is as de-*

fined in Lemma 2.2. Assume without loss of generality that $\|b\|_2 = 1$ *and take an initial guess* $x_0 = 0$. *Then the norm of the error at the kth step is bounded by*

$$\|e_k\|_2 \leq \kappa_2(A)\xi\epsilon^{(k)}. \tag{2.32}$$

Proof. This is simply a combination of Lemmas 2.2 and 2.3. □

We can derive an alternative bound under the same assumptions as Theorem 2.5

Theorem 2.6 *Assume that A is a non-singular and diagonalisable matrix of the form* $A = U\Lambda U^{-1}$. *Assume without loss of generality that* $\|b\|_2 = 1$ *and take the initial guess* $x_0 = 0$. *Then the norm of the error at the kth step is bounded by*

$$\|e_k\|_2 \leq \|A^{-1}\|_2\kappa_2(U)\epsilon^{(k)}. \tag{2.33}$$

Proof. By recalling (2.20) and the fact that $\|r_0\|_2 = \|b\|_2 = 1$, we have

$$\|r_k\|_2 \leq \kappa_2(U)\epsilon^{(k)}.$$

Application of (2.31) yields the required result. □

Both (2.32) and (2.33) involve constants that may be expensive to calculate, and the presence of $\kappa_2(A)$ and $\kappa_2(U)$ also implies that these bounds are of limited practical value. In a particular case, it may be difficult if not impossible to get accurate estimates for these values. In the work that follows, we will use the bound (2.32) on which to base subsequent theoretical results.

Once again we are faced with evaluating the quantity $\epsilon^{(k)}$. As in section 2.4 we will instead bound $\epsilon^{(k)}$ by $\eta^{(k)}$ on various complex domains D. We can use these expressions for $\eta^{(k)}$ to derive bounds on $\|e_k\|_2$ for the same specific distributions of $\sigma(A)$ as were considered in Theorem 2.4. In particular, if we take the case where $\sigma(A)$ is distributed on a disc, then we can state the following Theorem:

Theorem 2.7 *Assume that A is diagonalisable with eigenvalues distributed within a disc centre c and radius ρ not containing the origin. Further assume without loss of generality that $\|b\|_2 = 1$. Then the error at the kth step of the GMRES algorithm with an initial guess $x_0 = 0$ is bounded by*

$$\|e_k\|_2 \leq \kappa_2(A)\xi \left(\frac{\rho}{c}\right)^k, \tag{2.34}$$

where ξ is as defined in Lemma 2.2.

Proof. From Theorem 2.5 we have

$$\|e_k\|_2 \leq \kappa_2(A)\xi\epsilon^{(k)},$$

but for $\sigma(A) \in \{z : |z - c| \leq \rho, z \neq 0\}$ we have

$$\epsilon^{(k)} \leq \eta^{(k)} = \left(\frac{\rho}{c}\right)^k,$$

which completes the proof. □

Assuming we have good estimates for ρ and c, we are still left with the calculation of $\kappa_2(A)$ and ξ. As was mentioned in the discussion of Theorems 2.5 and 2.6, these calculations may be difficult. The following remark may be of more practical interest.

Remark 2.1 *Under the same assumptions as Theorem 2.7,*

$$\|e_k\|_2 = O\left(\frac{\rho}{c}\right)^k.$$

This follows immediately from Theorem 2.7. This result gives us some information on the rate of convergence even when we are unable to estimate the constants $\kappa_2(A)$ and ξ.

2.5.2 Test example

We can construct an example matrix where $\sigma(A)$ is distributed in a disc centred on 1 with a suitably small radius ρ. We can then calculate ξ and $\kappa_2(A)$ and if we choose an example where the exact solution is available, then we can calculate the error on each GMRES iteration. As the exact distribution of the spectrum is known, then we can calculate $\eta^{(k)}$ also.

Consider a matrix $A_\rho \in \mathbb{R}^{10\times 10}$ with eigenvalues uniformly distributed on a disc of radius $\rho = 0.1$ centred on $c = 1$. Thus we can calculate $\eta^{(k)} = (\rho/c)^k = 0.1^k$, and the condition number $\kappa(A) = 1.222$. Taking a right-hand side vector b with $\|b\|_2 = 1$, this allows us to calculate $\xi = 3.17$. With this data, we are now in a position to calculate the upper bound on $\|e_k\|_2$ given by (2.34) in Theorem 2.7. We can see from Table 2.1

k	1	2	3	4
$\kappa_2(A)\xi\eta^{(k)}$	3.874×10^{-1}	3.874×10^{-2}	3.874×10^{-3}	3.874×10^{-4}
$\|e_k\|_2$	1.009×10^{-1}	1.010×10^{-2}	1.010×10^{-3}	1.010×10^{-4}
k	5	6	7	8
$\kappa_2(A)\xi\eta^{(k)}$	3.874×10^{-5}	3.874×10^{-6}	3.874×10^{-7}	3.874×10^{-8}
$\|e_k\|_2$	1.010×10^{-5}	1.010×10^{-6}	1.010×10^{-7}	1.010×10^{-8}

Table 2.1: *Values for test matrix* A_ρ

that our bound (2.34) gives good agreement with observed results for this problem.

2.6 Convergence Behaviour III: Small Isolated Eigenvalues

2.6.1 A simple example

We now concern ourselves with the case where the majority of the spectrum is located in some complex region D bounded away from the origin, with a few eigenvalues situated near the origin. Figure 2-1 shows the distribution of the eigenvalues of matrix $A_{(\rho,s)}$ with $n = 20$. We have a single eigenvalue $\lambda_1 = 0.01$ with the rest of the eigenvalues $\lambda_i, i =$

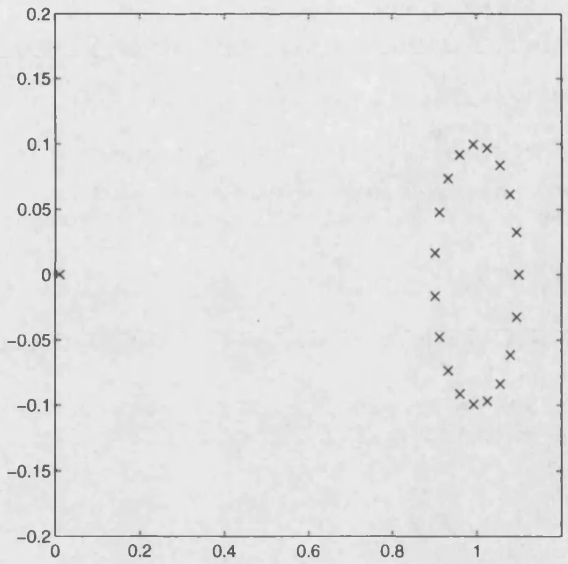

Figure 2-1: *Spectrum of example matrix* $A_{(\rho,s)}$

$2, \ldots, 20$ situated within a disc of radius $\rho = 0.1$ around 1. A straight application of Theorem 2.7 suggests poor convergence of GMRES. As this is an artificially constructed example, we are able to calculate the values of ξ and $\kappa_2(A)$, to obtain an accurate value for the right-hand side of inequality (2.34) for all k. The values of $\kappa_2(A)\xi(\rho/c)^k$ for $k = 1, \ldots, 10$ are shown in Table 2.2.

k	1	2	3	4	5
$\kappa_2(A)\xi(\rho/c)^k$	2.88×10^3	2.82×10^3	2.77×10^3	2.72×10^3	2.67×10^3
$\|e_k\|_2$	22.14	21.91	20.62	3.42	4.35×10^{-2}
k	6	7	8	9	10
$\kappa_2(A)\xi(\rho/c)^k$	2.63×10^3	2.58×10^3	2.53×10^3	2.49×10^3	2.44×10^3
$\|e_k\|_2$	1.05×10^{-3}	9.66×10^{-5}	9.65×10^{-6}	9.65×10^{-7}	9.65×10^{-8}

Table 2.2: *Values for* $\|e_k\|_2$ *for example and upper bound predicted by Theorem 2.7.*

We can see that Theorem 2.7 predicts slow convergence of the GMRES algorithm , but this is not what is observed in practice, as we obtain convergence to within reasonable accuracy in 10 iterations. From this example, we can see that the existing theory does not adequately describe the observed results when we have a small number of eigenvalues situated away from the rest of the spectrum.

The reason for the poor agreement with our initial bound (2.34) is the manner in which we are approximating the value of $\epsilon^{(k)}$. This quantity represents the smallest possible infinity norm that can be achieved by a polynomial of degree k over the set of points λ_i. Recalling equations (2.21) and (2.22), then we approximate $\epsilon^{(k)}$ by $\eta^{(k)}$, the infinity norm of a polynomial which has minimum value on a domain of which the λ_i are a subset. This approximation becomes inaccurate when the λ_i are distributed unevenly in the chosen domain. The polynomial that satisfies $\epsilon^{(k)}$ will be small on the points λ_i, but may have large values elsewhere in the domain. The polynomial that satisfies $\eta^{(k)}$ will have the smallest infinity norm over the whole domain for a fixed k, but may have significantly larger values at the λ_i than the polynomial that satisfies $\epsilon^{(k)}$. Thus the value computed for $\eta^{(k)}$ may be much larger than $\epsilon^{(k)}$, and so the rate of convergence predicted by Theorem 2.7 could be much slower than the observed rate.

When approximating $\epsilon^{(k)}$ by $\eta^{(k)}$, we calculate the value of $\eta^{(k)}$ using scaled and shifted Chebyshev polynomials. For a real domain D, we have the result

Theorem 2.8 *Let $0 < a < b$. The optimum*

$$\min_{p \in \mathcal{P}_k} \max_{t \in [a,b]} |p(t)|$$

is attained by

$$\hat{t}_k(t) = \frac{T_k[1 + 2(t-b)/(b-a)]}{T_k[1 - 2b/(b-a)]}, \tag{2.35}$$

and

$$\|\hat{t}_k\|_\infty = \frac{1}{T_k[1 - 2b/(b-a)]}. \tag{2.36}$$

Proof. See Rivlin [56]. □

If we consider a matrix whose eigenvalues lie on the real line in the domain

$$D_{\text{ideal}} = a \cup [b, c],$$

then currently we calculate $\eta^{(k)}$ using an optimal polynomial of the form (2.35) on the

domain

$$D = [a, c].$$

However, if we know the point a, then we can construct a polynomial that has zero value at this point:

$$\tilde{p}_k(t) = \left(1 - \frac{t}{a}\right) p_{k-1}, \tag{2.37}$$

where $p_{k-1} \in \mathcal{P}_{k-1}$. Since $\tilde{p}_k(0) = 1$, then $\tilde{p}_k \in \mathcal{P}_k$. Now

$$\min_{\tilde{p} \in \mathcal{P}_k} \max_{t \in D_{\text{ideal}}} |\tilde{p}(t)| = \min_{\tilde{p} \in \mathcal{P}_k} \max_{t \in [b,c]} |\tilde{p}(t)|$$

since by construction $\tilde{p}_k(a) = 0$. If we take p_{k-1} in (2.37) to be the optimal $(k-1)$th degree Chebyshev polynomial over $[b, c]$, then our value for $\eta^{(k)}$ will be more accurate than if we had used the optimal polynomial over $[a, c]$ - the polynomial can be as large as we like over (a, b) without affecting our estimate of $\epsilon^{(k)}$ (This type of approach is outlined with respect to semi-iterative methods by Hackbusch in [34]).

If we consider the example $A_{(\rho,s)}$ from §2.6.1 with $a = 0.1$, $b = 0.9$ and $c = 1.1$, then the optimal polynomial over $[a, c]$ is given by

$$\hat{t}_k(t) = \frac{T_k(1.2 - 2t)}{\cosh(k \cosh^{-1} 1.2)},$$

with

$$\|\hat{t}_k\|_\infty = \frac{1}{\cosh(k \cosh^{-1} 1.2)}$$

over the domain $D = [a, c]$. If we use a constructed polynomial of the form (2.37)

$$\tilde{p}_k(t) = (1 - 10t) \frac{T_{k-1}(10(t-1))}{\cosh(k \cosh^{-1} 10)},$$

then this has an infinity norm given by

$$\|\tilde{p}_k\|_\infty = \frac{10}{\cosh(k \cosh^{-1} 10)}$$

on our domain D_{ideal}. Comparing values of $\|\hat{t}_k\|_\infty$ and $\|\tilde{p}_k\|_\infty$ for various values of k on the respective domains, we find that the constructed polynomial $\tilde{p}_k$ yields a much smaller norm value for a given k. Figures are presented in Table 2.3.

k	2	4	6	8
$\|\hat{t}_k\|_\infty$	0.532	0.125	4.78×10^{-2}	1.14×10^{-2}
$\|\tilde{p}_k\|_\infty$	1.0	2.51×10^{-3}	6.33×10^{-6}	1.59×10^{-8}

Table 2.3: *Values of the infinity norm of the two polynomials $\hat{t}_k$ and $\tilde{p}_k$ over domains $D = [a, c]$ and D_{ideal} respectively, for varying values of k.*

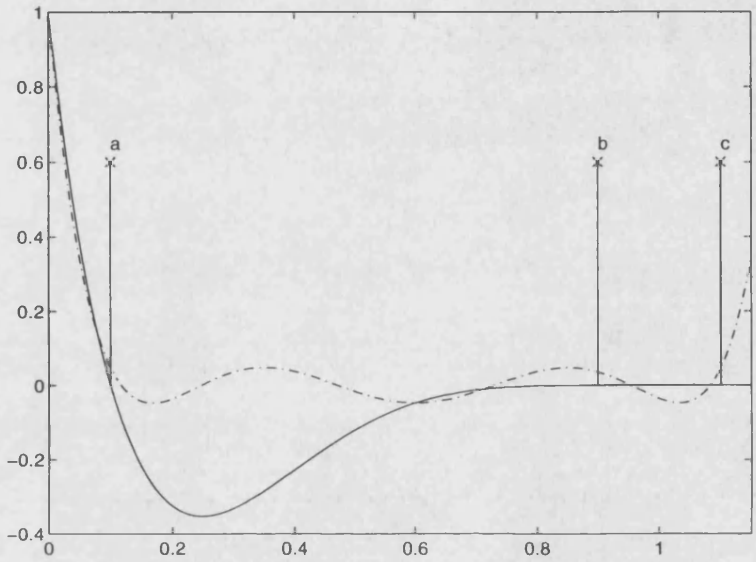

Figure 2-2: *Two 6th degree polynomials: The dash-dotted line is the optimal Chebyshev polynomial over $[a, c]$, the solid line is the polynomial constructed to be optimal over $[a] \cup [b, c]$.*

We can illustrate this example graphically - Figure 2-2 is a plot of two 6th degree polynomials. The dotted line is the optimal Chebyshev polynomial over $[0.1, 1.1]$ and the dot-dash line is our constructed polynomial (2.37). As can be seen, the optimal polynomial is smaller over the whole domain $[0.1, 1.1]$ but whilst the constructed polynomial has larger value on $(0.1, 0.9)$, it is much smaller than $\hat{t}_k$ on the ideal domain $\{[0.1] \cup [0.9, 1.1]\}$.

This approach can be generalised to a system where there exist more than one outlying eigenvalues. We will now present a result which allows us to approximate $\epsilon^{(k)}$ on the ideal domain, provided the outlying eigenvalues are known, by constructing a specific polynomial in the same manner as in (2.37).

Suppose that we have a nonsingular matrix $A \in \mathbb{R}^{n\times n}$ with a spectrum

$$\sigma(A) = \{\lambda_i,\, i = 1,\ldots,n,\, \lambda_1 < \lambda_2 < \ldots < \lambda_n\},$$

consisting of L small eigenvalues situated near the origin, with the remaining $n - L$ eigenvalues in some complex region D bounded away from the origin. So we have

$$\sigma(A) \in D_{\text{ideal}} = \left(\bigcup_{i=1}^{L} \lambda_i, \lambda_i \notin D\right) \cup \{z :\, z \in D\}. \tag{2.38}$$

We now define $\tilde{p} \in \mathbb{P}_k$

$$\tilde{p}(\lambda) = \prod_{i=1}^{L}\left(1 - \frac{\lambda}{\lambda_i}\right) p(\lambda), \tag{2.39}$$

where $p \in \mathcal{P}_{k-L}$. Since $\tilde{p}_k(0) = 1$, then $\tilde{p}_k \in \mathcal{P}_k$. We can use this polynomial to prove

Lemma 2.4 *Assume A is non-singular and diagonalisable, with eigenvalues $\lambda_i > 0$, $i = 1,\ldots,n$, $\lambda_1 < \lambda_2 < \ldots < \lambda_n$. Assume the first L of these eigenvalues are situated near the origin, with the rest bounded away from the origin in some complex region D. Define $\sigma_L = \{\lambda_i, i = 1,\ldots,L\}$, then*

$$\epsilon^{(k)} < |\lambda_n|^L \prod_{i=1}^{L} |\lambda_i^{-1}| \min_{p\in\mathcal{P}_{k-L}} \max_{\lambda\in\sigma(A)\setminus\sigma_L} |p(\lambda)|. \tag{2.40}$$

Proof. Recall from (2.19) that

$$\epsilon^{(k)} \equiv \min_{p\in\mathcal{P}_k} \max_{\lambda\in\sigma(A)} |p(\lambda)|.$$

Now clearly,

$$\min_{p\in\mathcal{P}_k} \max_{\lambda\in\sigma(A)} |p(\lambda)| \le \min_{\tilde{p}\in\mathcal{P}_k} \max_{\lambda\in\sigma(A)} |\tilde{p}(\lambda)|.$$

Considering

$$\min_{\tilde{p}\in\mathcal{P}_k} \max_{\lambda\in\sigma(A)} |\tilde{p}(\lambda)| = \min_{p\in\mathcal{P}_{k-L}} \max_{\lambda\in\sigma(A)} \left| p(\lambda) \prod_{i=1}^{L}\left(1 - \frac{\lambda}{\lambda_i}\right)\right|,$$

then we can say

$$\min_{p\in\mathcal{P}_{k-L}} \max_{\lambda\in\sigma(A)} \left| p(\lambda) \prod_{i=1}^{L}\left(1 - \frac{\lambda}{\lambda_i}\right)\right| = \min_{p\in\mathcal{P}_{k-L}} \max_{\lambda\in\sigma(A)\setminus\sigma_L} \left| p(\lambda) \prod_{i=1}^{L}\left(1 - \frac{\lambda}{\lambda_i}\right)\right|$$

since $\tilde{p}_k(\lambda) = 0$ when $\lambda \in \sigma_L$. Now

$$\begin{aligned}\min_{p\in\mathcal{P}_{k-L}} \max_{\lambda\in\sigma(A)\setminus\sigma_L} \left|p(\lambda)\prod_{i=1}^{L}\left(1-\frac{\lambda}{\lambda_i}\right)\right| &\leq \min_{p\in\mathcal{P}_{k-L}} \max_{\lambda\in\sigma(A)\setminus\sigma_L} |p(\lambda)|\prod_{i=1}^{L}\left|1-\frac{\lambda}{\lambda_i}\right| \\ &< \min_{p\in\mathcal{P}_{k-L}} \max_{\lambda\in\sigma(A)\setminus\sigma_L} |p(\lambda)|\prod_{i=1}^{L}\left|\frac{\lambda}{\lambda_i}\right| \\ &< \prod_{i=1}^{L}\frac{|\lambda_n|}{|\lambda_i|} \min_{p\in\mathcal{P}_{k-L}} \max_{\lambda\in\sigma(A)\setminus\sigma_L} |p(\lambda)|,\end{aligned}$$

which completes the proof (Note: Axelsson and Lindskog [6] use a polynomial similar to (2.39) to prove convergence results for the preconditioned CG method). □

We can use Lemma 2.4 to prove the following Theorem:

Theorem 2.9 *Assume A is non-singular and diagonalisable, with eigenvalues $\lambda_i > 0, i = 1,\ldots,n$, $|\lambda_1| < |\lambda_2| < \ldots < |\lambda_n|$. Assume the first L of these eigenvalues are situated near the origin, with the rest bounded away from the origin in a disc centred on c with radius ρ, so*

$$\sigma(A) \in \left(\bigcup_{i=1}^{L}\lambda_i\right) \cup \{z : |z-c| \leq \rho, z \neq 0\}, \lambda_L < c-\rho. \tag{2.41}$$

Define

$$\bar{\eta}^{(k)} = (c+\rho)^L \prod_{i=1}^{L}|\lambda_i^{-1}|\left(\frac{\rho}{c}\right)^{k-L},$$

then the error at the kth step of the GMRES algorithm applied to to the linear system $Ax = b$, assuming without loss of generality that $\|b\|_2 = 1$, with initial guess $x_0 = 0$ is bounded by

$$\|e_k\|_2 < \kappa_2(A)\xi\bar{\eta}^{(k)}, \tag{2.42}$$

where ξ is as defined in Lemma 2.2.

Proof. Lemma 2.2 gives us

$$d_k \leq \xi\epsilon^{(k)},$$

and by Lemma 2.4 we have

$$d_k < \xi|\lambda_n|^L \prod_{i=1}^{L} |\lambda_i^{-1}| \min_{p\in\mathcal{P}_{k-L}} \max_{\lambda\in\sigma(A)\setminus\sigma_L} |p(\lambda)|.$$

As λ_i, $i = L+1,\ldots,n$ are contained in the disc $\{z : z-c \leq \rho, z \neq 0\}$, then $|\lambda_n| \leq c+\rho$, and applying Theorem 2.4(b) gives us

$$\min_{p\in\mathcal{P}_{k-L}} \max_{\lambda\in\sigma(A)\setminus\sigma_L} |p(\lambda)| \leq \left(\frac{\rho}{c}\right)^{k-L}.$$

Finally, applying Lemma 2.3 and gives the required result. □

We can now observe

Remark 2.2 *Under the same assumptions as Theorem 2.9,*

$$\|e_k\|_2 = O\left(\frac{\rho}{c}\right)^k.$$

Similarly to Remark 2.1, this result gives us a qualitative estimate on the rate of convergence in cases where we are unable to calculate constants such as ξ and $\kappa_2(A)$, provided we have some information about the distribution of $\sigma(A)$. We can now return to example $A_{(\rho,s)}$ to see if our new bound gives a more accurate idea of the rate of convergence. Table 2.4 shows the results. We can see from the results that although the bound is not

k	1	2	3	4	5
$\kappa_2(A)\xi\bar{\eta}^{(k)}$	3.22×10^5	3.22×10^4	3.22×10^3	3.22×10^2	3.22×10^1
$\|e_k\|_2$	22.14	21.91	20.62	3.42	4.35×10^{-2}
k	6	7	8	9	10
$\kappa_2(A)\xi\bar{\eta}^{(k)}$	3.22	3.22×10^{-1}	3.22×10^{-2}	3.22×10^{-3}	3.22×10^{-4}
$\|e_k\|_2$	1.05×10^{-3}	9.66×10^{-5}	9.65×10^{-6}	9.65×10^{-7}	9.65×10^{-8}

Table 2.4: *Values for $\|e_k\|_2$ for example and upper bound predicted by Theorem 2.9.*

sharp, it still represents a large improvement over the bound given by Theorem 2.7.

The theorems and results presented in this section have shown that it is possible to derive reasonable accurate bounds for the error produced by GMRES when a matrix has eigenvalues situated near the origin, away from the remainder of the spectrum. The

bounds have shown that we can expect a similar rate of convergence for these problems as we would obtain if the outlying eigenvalues were absent. However, they still have implications for the accuracy of an approximate solution when GMRES is terminated using the norm of the residual as a convergence criterion. This will be discussed in Chapters 3 and 4.

2.6.2 Large isolated eigenvalues

We will briefly discuss here the effect that large outlying eigenvalues on the convergence of the algorithm. We can easily construct a matrix with the same condition number as $A_{(\rho,s)}$ but with a single large outlying eigenvalue in place of the small outlier. We will denote this matrix by $A_{(\rho,l)}$. If we recall the constructed polynomial (2.39), then we can perform a similar analysis to that in Lemma 2.4 to prove

Lemma 2.5 *Assume A is non-singular and diagonalisable, with eigenvalues $\lambda_i > 0$, $i = 1,\ldots,n$, $\lambda_1 < \lambda_2 < \ldots < \lambda_n$. Assume the first $n-L$ of these eigenvalues are bounded away from the origin in some complex region D, with the last L situated to the right of D. Define $\sigma_L = \{\lambda_i, i = n-L+1,\ldots,n\}$, then*

$$\epsilon^{(k)} < \min_{p\in\mathcal{P}_{k-L}} \max_{\lambda\in\sigma(A)\setminus\sigma_L} |p(\lambda)|. \tag{2.43}$$

Proof. From the proof of Lemma 2.4, we have

$$\begin{aligned}
\epsilon^{(k)} &\leq \min_{p\in\mathcal{P}_{k-L}} \max_{\lambda\in\sigma(A)\setminus\sigma_L} |p(\lambda)| \prod_{i=n-L+1}^{n} \left|1 - \frac{\lambda}{\lambda_i}\right| \\
&= max_{\lambda\in\sigma(A)\setminus\sigma_L}|p(\lambda)| \prod_{i=n-L+1}^{n} \left|\frac{\lambda_i - \lambda}{\lambda_i}\right| \\
&< \prod_{i=n-L+1}^{n} \left|\frac{\lambda_i - \lambda_1}{\lambda_i}\right| \min_{p\in\mathcal{P}_{k-L}} \max_{\lambda\in\sigma(A)\setminus\sigma_L} |p(\lambda)| \\
&< \prod_{i=n-L+1}^{n} \left|\frac{\lambda_i}{\lambda_i}\right| \min_{p\in\mathcal{P}_{k-L}} \max_{\lambda\in\sigma(A)\setminus\sigma_L} |p(\lambda)|
\end{aligned}$$

□

Comparing (2.40) with (2.43), then we can see that the bounds are identical apart from the product term in front of the minimax polynomial in (2.40). For the small eigenvalue case, this term will be greater than one, and if the eigenvalues are very small, much greater than one. Conversely, for the large eigenvalue case this term is absent. This, in conjunction with Theorem 2.5 and Lemma 2.2, suggests that the rate of convergence will be similar for examples $A_{(\rho,s)}$ and $A_{(\rho,l)}$ (from Remark 2.1) but the error with matrix $A_{(\rho,s)}$ at the kth step will be larger than that with matrix $A_{(\rho,l)}$. This is illustrated in Figure 2-3, which shows plots of the log of the error for both $A_{(\rho,s)}$ and $A_{(\rho,l)}$. Thus we can expect matrices with larger isolated eigenvalues to converge in fewer iterations than similarly conditioned matrices with small isolated eigenvalues, although the asymptotic rate of convergence will be similar.

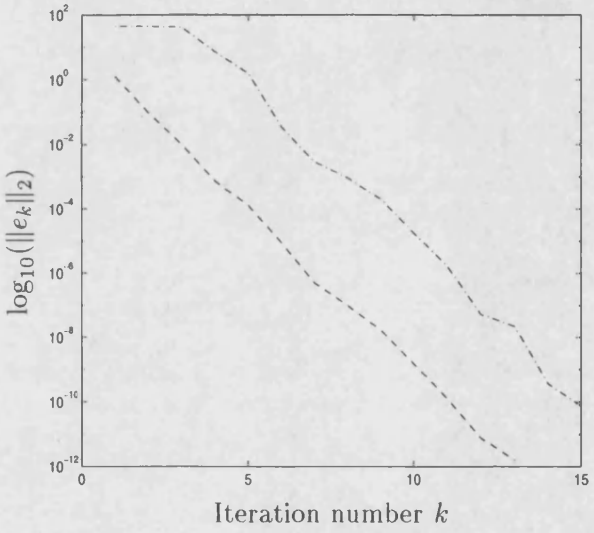

Figure 2-3: *Plots of* $\log_{10}(e_k)$ *for GMRES applied to* $A_{(\rho,s)}x = b$ *(dotted line) and* $A_{(\rho,l)}x = b$ *(dot-dashed line) with* $b = (1, 1, \ldots, 1)^T$.

2.6.3 Further Analysis

We now return to the small eigenvalue case to re-examine the improved bound (2.42). Although (2.42) represents an improvement over the initial bound (2.34) for matrices

such as $A_{(\rho,s)}$ that possess isolated eigenvalues, it is still not very accurate for these cases. If we examine the bound obtained for d_k for the case where the spectrum $\sigma(A)$ is distributed in the form (2.41),

$$d_k < \xi\bar{\eta}^{(k)}, \tag{2.44}$$

then experiments with example $A_{(\rho,s)}$ show that this bound is quite tight. Figure 2-4 is a plot of the log of d_k and the log of the above bound for the matrix $A_{(\rho,s)}$. This clearly shows the good agreement of the bound with the quantity d_k. This implies that it is

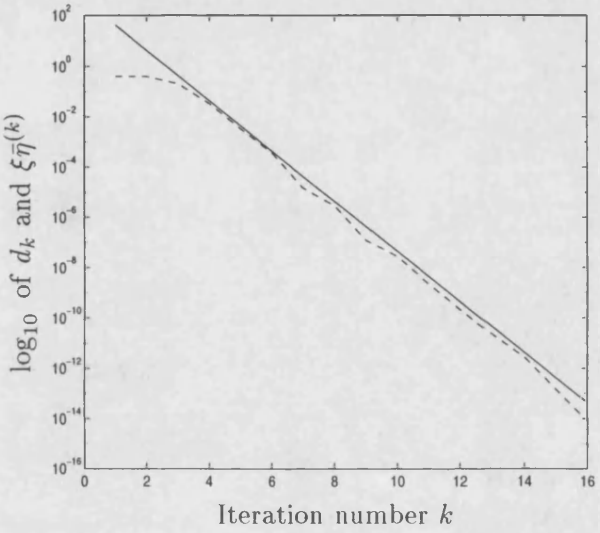

Figure 2-4: *Plots of* $\log_{10}(d_k)$ *(dashed line) and* $\log_{10}(\xi\bar{\eta}^{(k)})$ *(solid line) for example* $A_{(\rho,s)}$.

the use of the inequality (2.29),

$$\|e_k\|_2 \leq \kappa_2(A)d_k$$

(Lemma 2.3), in the proof of Theorem 2.5 that is making the bound inaccurate. We therefore require an improvement over the existing error bound (2.29) for $\|e_k\|_2$ that more accurately describes the influence of the small eigenvalue on the behaviour of the error. If we consider the case where we have one isolated eigenvalue and expand the kth

residual in terms of the eigenbasis of A in the form

$$r_k = \sum_{i=1}^{n} \rho_i^{(k)} u_i,$$

then recalling (2.31) that

$$e_k = A^{-1} r_k$$

allows us to write

$$e_k = \sum_{i=1}^{n} \frac{\rho_i^{(k)}}{\lambda_i} u_i. \tag{2.45}$$

If we now split the sum to extract the term involving the small eigenvalue, taking norms gives us

$$\begin{aligned} \|e_k\|_2 &= \|\frac{\rho_1^{(k)}}{\lambda_1} u_1 + \sum_{i=2}^{n} \frac{\rho_i^{(k)}}{\lambda_i} u_i\|_2 \\ &\leq \frac{|\rho_1^{(k)}|}{|\lambda_1|} + \sum_{i=2}^{n} \frac{|\rho_i^{(k)}|}{|\lambda_i|} \\ &\leq \frac{|\rho_1^{(k)}|}{|\lambda_1|} + \frac{\|r_k\|_2}{|\lambda_2|}. \end{aligned}$$

We can now state

Theorem 2.10 *Under the same assumptions as Theorem 2.9, the error at the kth step of the GMRES algorithm applied to the system $Ax = b$ is bounded by*

$$\|e_k\|_2 \leq \sum_{i=1}^{L} \frac{|\rho_i^{(k)}|}{|\lambda_i|} + \frac{\|A\|_2 \kappa(A)}{|\lambda_{L+1}|} d_k, \tag{2.46}$$

where $r_k = \sum_{i=1}^{n} \rho_i^{(k)} u_i$.

Proof. We can extend the process performed above to obtain

$$\|e_k\|_2 \leq \sum_{i=1}^{L} \frac{|\rho_i^{(k)}|}{|\lambda_i|} + \frac{\|r_k\|_2}{|\lambda_{L+1}|},$$

and the inequality

$$\|r_k\|_2 \leq \|A\|_2 \kappa(A) d_k$$

completes the result. □

Bound (2.46) is not as neat as previous results as the first term cannot be calculated *a priori* due to the presence of the coefficients $\rho_i^{(k)}$. However, if we consider the residual

$$r_k = b - Ax_k = \sum_{i=1}^{n} (\beta_i - \lambda_i \alpha_i^{(k)}) u_i,$$

where x_k has been expanded in the eigenbasis of A in the form $x_k = \sum_{i=1}^{n} \alpha_i^{(k)} u_i$, then

$$\rho_i^{(k)} = \beta_i - \lambda_i \alpha_i^{(k)}.$$

However, we know (2.28) that $\beta_i = \lambda_i \alpha_i$, so

$$\rho_i^{(k)} = \lambda_i (\alpha_i - \alpha_i^{(k)}).$$

So the behaviour of the coefficients $\rho_i^{(k)}$ is governed by the convergence of $\alpha_i^{(k)}$ to α_i. Recalling that we take the initial guess for GMRES to be $x_0 = 0$, then we can expand the kth iterate in terms of the eigenbasis formed from the Ritzvectors of H_k, to obtain

$$\sum_{i=1}^{n} \alpha_i^{(k)} u_i = \sum_{j=1}^{k} \gamma_j^{(k)} \mathsf{v}_j,$$

where $\mathsf{v}_j = V_k \mathsf{u}_j$ with $\{\mathsf{u}_j, j = 1, \ldots, k\}$ being the Ritzvectors of H_k. If we consider an eigenvector u_i corresponding to one of the L smallest eigenvalues, then as the GMRES iteration proceeds, the vector v_i converges to the vector u_i, so $\gamma_i^{(k)}$ converges to $\alpha_i^{(k)}$. By this loose heuristical argument, we can say that

$$\gamma_i^{(k)} \to \alpha_i^{(k)} \to \alpha_i \text{ as } k \to n.$$

Thus it may be possible to relate the behaviour of the coefficients $\rho_i^{(k)}$ to the convergence of the approximate eigenvectors obtained from the Ritz vectors of the matrix H_k: If v_i converges to u_i rapidly, then $\gamma_i^{(k)}$ should converge to α_i rapidly, and the corresponding term $\rho_i^{(k)}$ would become small. One way to measure the convergence of the approximate eigenvectors is to consider the distance $\|(I - \pi_k)u_i\|_2$ in a similar manner to that performed in section 2.5.1 for the distance d_k (2.26) (see Chatelin [14] and Saad [59] for details). This avenue is a candidate for future work, and is not considered any further

in this Thesis.

We can now return to example $A_{(\rho,s)}$ with this new bound (2.46). If we calculate the eigenbasis $U = [u_i, i = 1, \ldots, n]$ of A, then we are able to calculate the coefficients $\rho^{(k)} = (\rho_i^{(k)}, i = 1, \ldots, n)^T$ by solving the linear system

$$U^T U \rho^{(k)} = U^T r_k.$$

We can use the expression (2.44) to bound d_k, and calculate the quantity $\frac{|\rho_1^{(k)}|}{|\lambda_1|}$ exactly. The results are shown in Figure 2-5. As can be seen, this bound is quite tight after a few

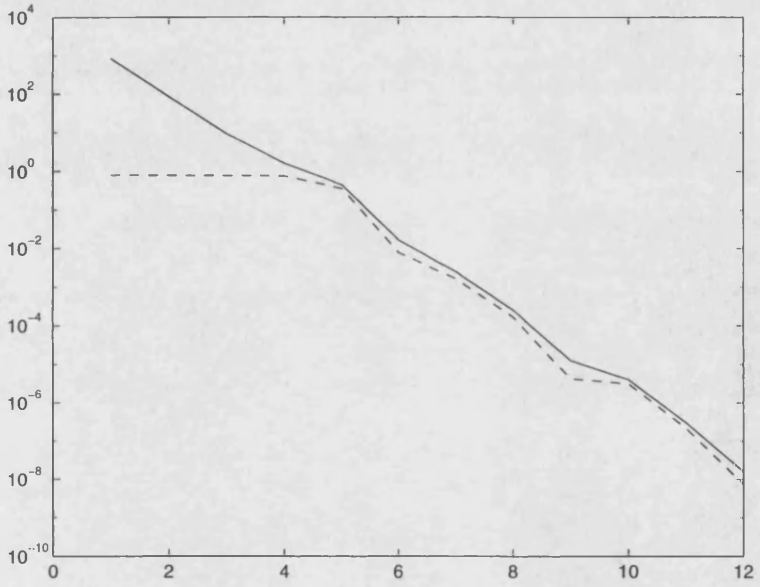

Figure 2-5: *Plots of* $\log_{10}(\|e_k\|_2)$ *(dashed line) and* $\log_{10}\left(\frac{|\rho_1^{(k)}|}{|\lambda_1|} + \xi\bar{\eta}^{(k)}\frac{\|A\|_2\kappa(A)}{|\lambda_2|}\right)$ *(solid line) for the matrix* $A_{(\rho,s)}$.

iterations, representing a marked improvement even over the improved bound (2.42) for the case when the matrix has several small eigenvalues situated away from the rest of the spectrum. However, it has the drawback that it is not computable *a priori* due to the $\rho_i^{(k)}$ terms. Table 2.5 shows the values of $\|e_k\|_2$, $\frac{|\rho_1^{(k)}|}{|\lambda_1|}$ and $\xi\bar{\eta}^{(k)}\frac{\|A\|_2\kappa(A)}{|\lambda_2|}$ for example $A_{(\rho,s)}$. From this, it is clear that the term $\rho_i^{(k)}$ plays a major role in the convergence of the algorithm, being the larger of the two terms for much of the iteration. As with previous results, our new bound (2.46) is of little practical use due to the difficulty of calculating

k	1	2	3	4
$\|e_k\|_2$	7.974×10^{-1}	7.832×10^{-1}	7.676×10^{-1}	7.443×10^{-1}
$\frac{\|\rho_1^{(k)}\|}{\|\lambda_1\|}$	7.930×10^{-1}	7.857×10^{-1}	7.736×10^{-1}	7.500×10^{-1}
$\xi\bar{\eta}^{(k)}\frac{\|A\|_2\kappa(A)}{\|\lambda_2\|}$	8.403×10^{2}	8.403×10^{1}	8.403×10^{0}	8.403×10^{-1}
k	5	6	7	8
$\|e_k\|_2$	3.545×10^{-1}	7.772×10^{-3}	1.612×10^{-3}	1.677×10^{-4}
$\frac{\|\rho_1^{(k)}\|}{\|\lambda_1\|}$	3.574×10^{-1}	7.903×10^{-3}	1.626×10^{-3}	1.690×10^{-4}
$\xi\bar{\eta}^{(k)}\frac{\|A\|_2\kappa(A)}{\|\lambda_2\|}$	8.403×10^{-2}	8.403×10^{-3}	8.403×10^{-4}	8.403×10^{-5}
k	9	10	11	12
$\|e_k\|_2$	4.182×10^{-6}	3.116×10^{-6}	2.169×10^{-7}	7.372×10^{-9}
$\frac{\|\rho_1^{(k)}\|}{\|\lambda_1\|}$	4.096×10^{-6}	3.130×10^{-6}	2.198×10^{-7}	7.307×10^{-9}
$\xi\bar{\eta}^{(k)}\frac{\|A\|_2}{\|\lambda_2\|}$	6.876×10^{-6}	6.876×10^{-7}	6.876×10^{-8}	6.876×10^{-9}

Table 2.5: *Values of* $\|e_k\|_2$, $\frac{|\rho_1^{(k)}|}{|\lambda_1|}$ *and* $\xi\bar{\eta}^{(k)}\frac{\|A\|_2}{|\lambda_2|}$ *for GMRES applied to test matrix* $A_{(\rho,s)}$

the constants required for its evaluation, and a result such as Corollary 2.2 may be of more use when applied to realistic problems. Still, the derivation of (2.46) has illustrated the importance of small eigenvalues in the rate of convergence of the GMRES algorithm (note that an identical analysis could be performed with large outlying eigenvalues. However, the $\frac{|\rho_L^{(k)}|}{|\lambda_L|}$ term would be small and so have little contribution to the error).

2.7 Problems with the GMRES algorithm

2.7.1 Loss of orthogonality

Whilst in exact arithmetic the matrix V_k (2.7), whose columns are the basis vectors v_i for the Krylov subspace $\mathcal{K}_k(A, r_0)$, is l_2-orthonormal, in finite precision arithmetic this is not always true. The original Saad and Schultz implementation of the GMRES algorithm [61] used the modified Gramm-Schmidt process (MGS) [31] to construct the basis vectors v_i. Whilst this process possesses superior numerical properties than the standard Gramm-Schmidt process, it is still liable to produce vectors with some degree of linear dependence. Bjork [8] showed that if MGS is applied to a set of vectors $S = \{v_1, \ldots, v_k\}$ producing the set of vectors $Q = \{q_1, \ldots, q_k\}$ using finite precision

arithmetic with unit rounding error $\mathbf{u}$, then

$$Q^T Q = I + E, \tag{2.47}$$

where I is the $k \times k$ identity matrix and

$$\|E\|_2 \approx \mathbf{u}\kappa_2(S),$$

where $\kappa_2(S)$ is the ratio of the largest to the smallest singular values of S. Thus if the original set of vectors S to be orthogonalised is sufficiently ill-conditioned, then the resulting set of vectors Q may suffer from significant loss of orthogonality. This means that at any stage of the GMRES algorithm implemented with MGS, the most recent basis vector v_{k+1} may have a significant non-zero component in span$\{v_1, \ldots, v_k\}$.

Several strategies have been suggested to combat the loss of orthogonality in the basis vectors. The simplest approach is to reorthogonalise the vectors. Once the new basis vector v_{k+1} has been formed, but before normalisation, it has the Gramm-Schmidt process applied to it again. The coefficients in the $\bar{H}_k$ matrix are updated during this process.

Algorithm 2.4 *Reorthogonalisation*

1. **Start**: Vector v_{k+1} from modified Gramm-Schmidt process
2. **Iterate**: For $j = 1, \ldots, k$:
   $h_{new} = (v_{k+1}, v_j)$,
   $h_{j,k} = h_{j,k} + h_{new}$,
   $v_{k+1} = v_{k+1} - h_{new} v_j$.
3. **End**: Update last row of $\bar{H}_k$:
   $h_{k+1,k} = \|v_{k+1}\|_2$.

Clearly, this approach is very expensive, doubling the cost of the orthogonalisation process. The above algorithm takes no account of whether the basis vectors are suffering from loss of orthogonality or not - the reorthogonalisation takes place regardless. Parlett [53] shows that little is gained from subsequent reorthogonalisations; one application of Algorithm 2.4 is sufficient.

2.7.2 Detecting loss of orthogonality

It is possible to check at each step of the MGS process for loss of orthogonality without performing additional inner-products. Brown and Hindmarsh [10] suggest a variant of a method from [53]. Rounding errors begin to occur if the new direction Av_k to be orthogonalised is very close to one of the previous directions $\{v_k\}$. If this occurs, then the inner product $h_{i,k} = (Av_k, v_i)$ will be very close to $\|Av_k\|_2$ for the corresponding direction v_i. The resulting corrected vector v_{k+1} will then have a much smaller magnitude than the original direction Av_k after subtraction of the component $h_{i,k} v_i$. Brown and Hindmarsh consider the quantity

$$\|Av_k\|_2 + \alpha \|v_{k+1}\|_2 \tag{2.48}$$

where α is some small parameter (typically $\alpha = 0.001$). This quantity is compared to $\|Av_k\|_2$, and if they are equal to within machine precision, then Algorithm 2.4 is performed. Such a selection method saves the expense of performing unrequired reorthogonalisations, which can require large numbers of calculations if k is large.

2.7.3 Householder GMRES

Walker [71] proposes an alternative implementation of the GMRES algorithm using Householder transformations to produce the basis vectors for $\mathcal{K}_k$.

If we have a non-zero vector $v \in \mathbb{R}^n$, then the $n \times n$ matrix P of the form

$$P = I - 2vv^T/v^Tv \tag{2.49}$$

is called a Householder transformation. An equivalent definition is

$$P = I - 2uu^T$$

subject to $\|u\|_2 = 1$. Such matrices P are symmetric and orthogonal, and have the important property of being able to zero specific entries in a vector. In particular, given a vector $x \in \mathbb{R}^n$, then it is easy to construct a u such that Px is a multiple of e_1, the first column of I. If we consider

$$Px = (I - 2uu^T)x = x - 2u^Txu,$$

then $Px \in \text{span}\{e_1\}$ implies $u \in \text{span}\{e_1, x\}$. If we set

$$u = \gamma(x + \beta e_1)$$

with

$$\gamma = (x^Tx + 2\beta x_1 + \beta^2)^{-1/2}$$

where $x_1 = x \cdot e_1$, then our expression for Px becomes

$$Px = (1 - 2\gamma^2(x^Tx + \beta x_1))x - 2\beta\gamma^2(x^Tx + \beta x_1)e_1.$$

Thus the requirement that $Px = \alpha e_1$ is satisfied if we set $\beta = \pm\|x\|_2$. Note from the above example that the action of a Householder transformation on a vector can be calculated using only the Householder vector - there is no need to explicitly form the transformation matrix.

2.7.4 Householder orthonormalisation

Householder transformations can be used to orthonormalise a set of vectors. To orthonormalise the columns of $S = \{s_1, \ldots, s_k\}$, we simply determine Householder transformations $P_1, \ldots, P_k$ such that $P_k \ldots P_1 S = R$, where R is an upper triangular matrix. Since $S = P_1 \ldots P_k R$, the matrix Q formed from the first k columns of $P_1 \ldots P_k$ is our orthonormal basis for S (for full details of the orthonormalisation process, see Golub and Van Loan [31]).

Bjorck [8] showed that using floating-point arithmetic with unit round-off $\mathbf{u}$, then

$$Q^T Q = I + E, \tag{2.50}$$

where I is again the $k \times k$ identity matrix, and

$$\|E\|_2 \approx \mathbf{u}$$

when Q is calculated using Householder orthonormalisation. Thus Householder orthogonalisation is numerically more reliable than the modified Gramm-Schmidt process where $\|E\|_2 \approx \mathbf{u}\kappa_2(S)$. Due to this reason, Walker proposed the implementation of GMRES [71] using the Householder orthonormalisation process in place of the MGS process. This implementation requires less storage than the original implementation [61] but requires more arithmetic. The decrease in storage occurs as the basis vectors $\{v_k\}$ are not stored explicitly. Instead, the Householder vectors u_k which determine the transformation matrices P_k are stored, and the vectors v_k are calculated as required. The additional arithmetic arises at the start of the orthonormalisation process with the formation of the new basis vector v_{k+1}. Comparing the amount of work required by the two iterations, it can be shown [71] that the Householder implementation requires $n(3k^2 + 2k + 2)$ multiplication and $k+1$ matrix-vector products with A to perform k iterations and compute the approximate solution x_k, whereas the MGS implementation with no reorthogonalisation requires $n(k^2 + 4k + 2)$ multiplications and $k + 1$ matrix-vector products with A. If reorthogonalisation is carried out on each iteration, then $n(2k^2 + 5k + 2)$ multiplications are required. So even with the application of Algorithm 2.4 at every step, the MGS implementation is cheaper than the Householder version. Walker indicates

that the additional arithmetic might not be a prohibitive factor if the restart factor m (Algorithm 2.3) is small, or in cases where evaluating the matrix-vector products with A are expensive. Walker presents several examples which show the improved reliability of the Householder orthogonalisation process over the MGS process.

We have outlined above some practical strategies to combat loss of orthogonality in GMRES, which will be used in later Chapters.

2.8 Summary

In this Chapter we have presented some results that describe the convergence behaviour of the GMRES algorithm, and illustrated them with some simple examples. We have also mentioned some ideas to combat the effects of finite-precision arithmetic when the method is applied to ill-conditioned systems. In the next Chapter, we will consider some linear systems from SPEEDUP and examine the concept of preconditioning the GMRES algorithm to aid the rate of convergence.

Chapter 3

Preconditioners

3.1 Introduction

In this chapter, we discuss preconditioning methods for GMRES. We present several examples arising from SPEEDUP problems that demonstrate the need for preconditioning, and examine the incomplete LU factorisation (ILU), a popular preconditioning method for non-symmetric linear systems. The poor performance of ILU preconditioned GMRES for these examples is discussed, and an alternative preconditioning method is introduced, based on complete LU factorisation. This utilises the fact that we are solving a *sequence* of linear systems (1.1),

$$J_l x = b_l, \quad l = 1, 2, \ldots$$

where the J_l are Jacobian matrices arising from the solution of nonlinear systems $f(x) = 0$ by some variant of Newton's method (see §1.4.1). Firstly, we will motivate the need to precondition iterative methods (in our case, GMRES) for solving linear systems

$$Ax = b$$

when the matrix A is from a 'real world' problem.

3.2 Why precondition?

We have seen from chapter 2 that provided a matrix A has a suitably distributed spectrum, then the GMRES algorithm will converge at a 'satisfactory' rate. We saw in Chapter 2 that if the majority of the eigenvalues of a matrix are grouped together reasonably tightly, we can expect good convergence of GMRES. Unfortunately, matrices from realistic problems rarely possess such a distribution (see, for example, Figures 3-2, 3-4 and 3-6). By preconditioning the problem, we hope to improve the distribution of the spectrum (i.e. make the eigenvalues more tightly clustered) and hence increase the rate of convergence of the algorithm. This process consists of calculating a *preconditioner* M, and solving the preconditioned system

$$M^{-1}Ax = M^{-1}b. \tag{3.1}$$

This is known as left preconditioning; right preconditioning is also possible:

$$\begin{aligned} AM^{-1}y &= b, \\ x &= M^{-1}y. \end{aligned}$$

For symmetric problems, a combination of left and right preconditioning is usually required to preserve symmetry, but for the non-symmetric SPEEDUP problems we will concentrate on left preconditioning. This has the effect of skewing the residual norm in a potentially advantageous manner. Whereas right preconditioning leaves the residual norm unaffected, left preconditioned GMRES yields estimates of the norm of the preconditioned residual. We have

$$M^{-1}r_k = M^{-1}b - M^{-1}Ax_k = M^{-1}Ax - M^{-1}Ax_k = M^{-1}Ae_k,$$

so if our preconditioner M is 'close' to A, then the preconditioned residual norm will be a good estimate of the error norm:

$$\|M^{-1}Ae_k\|_2 \simeq \|e_k\|_2,$$

since $M^{-1}A$ will be 'close' to the identity matrix. Note that as in most preconditioned iterative schemes, the preconditioned matrix $M^{-1}A$ is never formed explicitly; the action of it on a vector is calculated in two stages, first a matrix-vector multiplication with A and secondly the solution of a linear system

$$Mz = y.$$

A preconditioned system must satisfy two requirements:

- The eigenvalues of the preconditioned system should be distributed to aid faster convergence than that of the original system. From Chapter 2, we know for GMRES that if the majority of the eigenvalues are tightly clustered away from zero, then convergence will be rapid.
- the preconditioner is easy to apply, i.e. $My = z$ is cheap to solve.

Clearly, there is no such thing as an 'ideal' preconditioner: We cannot satisfy both the above conditions exactly. For example, if we were to take $M = A$, then we would get immediate convergence of the GMRES algorithm, but at the cost of having to solve the original system during the iteration process. Conversely, taking $M = I$ means that the linear system $My = z$ is solved trivially but convergence is not altered at all. What is required is a compromise: A preconditioner that is not too expensive to calculate or apply, but which yields a worthwhile increase in the convergence of the algorithm, so that the overall solution time is less than that required for the unpreconditioned system.

What follows is a description of four example matrices from SPEEDUP models that will be used to test the effectiveness of the various preconditioners that are presented later in the chapter, along with results of numerical experiments carried out with these examples.

3.3 Examples

In this section, we give a brief description of the matrices from SPEEDUP on which we will test the performance of the various preconditioners. All these matrices are extremely poorly scaled: For example, in Example 1 the largest absolute value of an element in an example Jacobian matrix is 2.92×10^6, and the smallest is 1.93×10^{-6}. Scaling of the Jacobians is not performed unless scale factors are explicitly prescribed by the authors of the flowsheet. For the problems presented below, no scale factors were given. We present sparsity patterns of sample Jacobian matrices, along with eigenvalues calculated numerically using MATLAB. This uses a QR factorisation (see, for example, Golub and Van Loan [31]) of the matrix to compute the eigenvalues. Note that rounding error means that the eigenvalues presented here are unlikely to be the exact eigenvalues of the matrices, but only approximations to them.

3.3.1 Example 1: BTX separation column

The first example we will consider is from the largest nonlinear block of a SPEEDUP demonstration model (more details of this model are given in §4.4). The sparsity pattern of a typical Jacobian matrix is shown in Figure 3-1, and an unpreconditioned spectrum is shown in Figure 3-2. This figure shows six frames, with the entire spectrum shown in the top left corner, with the remaining five frames showing increasing degrees of magnification. The dimension of the matrix is $n = 927$ with a condition number of $\kappa_2(A) = 7.01 \times 10^{11}$. If we define the density of a sparse matrix;

Definition 3.1 *The density of a sparse matrix $A \in \mathbb{R}^{n \times n}$ is given by*

$$\mathcal{D} = \frac{\mathrm{NZ}}{n^2},$$

where NZ is the number of non-zero elements in A,

then the matrices from this example have a density of $\mathcal{D} = 4.8 \times 10^{-3}$.

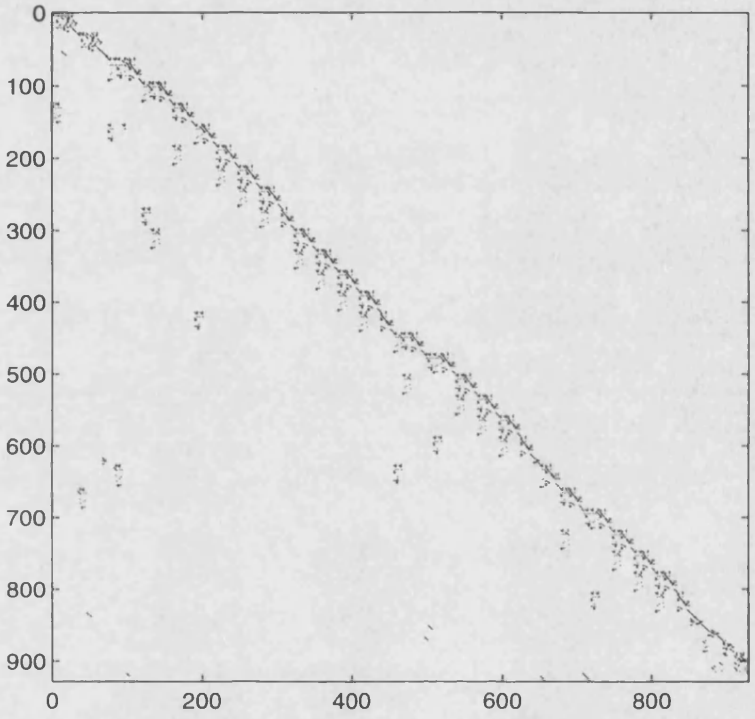

Figure 3-1: *Sparsity pattern of a typical Example 1 matrix. The matrix is of size* $n = 927$ *with 4119 non-zero entries.*

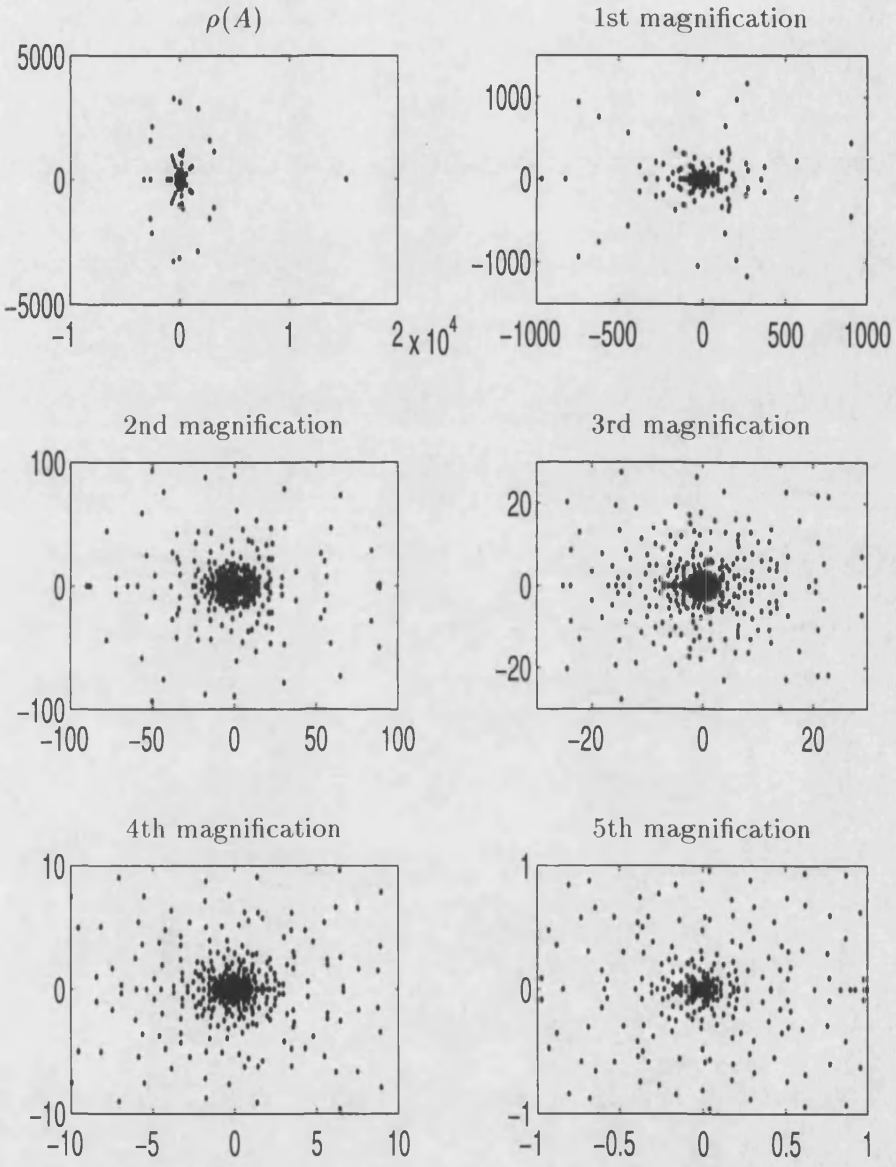

Figure 3-2: *Spectrum of typical Example 1 matrix. Top left is whole spectrum, followed by increasing degrees of magnification.*

3.3.2 Example 2: Plant 1 block 1

The second example comes from the second largest nonlinear block of a chemical plant model we shall call Plant 1. For reasons of confidentiality, no details of the model can be given. The matrices are of size $n = 719$, with 2554 non-zero entries. The sparsity pattern of a typical matrix is shown in Figure 3-3. This matrix has a condition number of $\kappa_2(A) = 4.87 \times 10^{16}$, so despite being smaller than the matrix in Example 1, possesses more extreme conditioning. Figure 3-4 shows the spectrum of an example Jacobian. The matrix has a density of $\mathcal{D} = 4.9 \times 10^{-3}$.

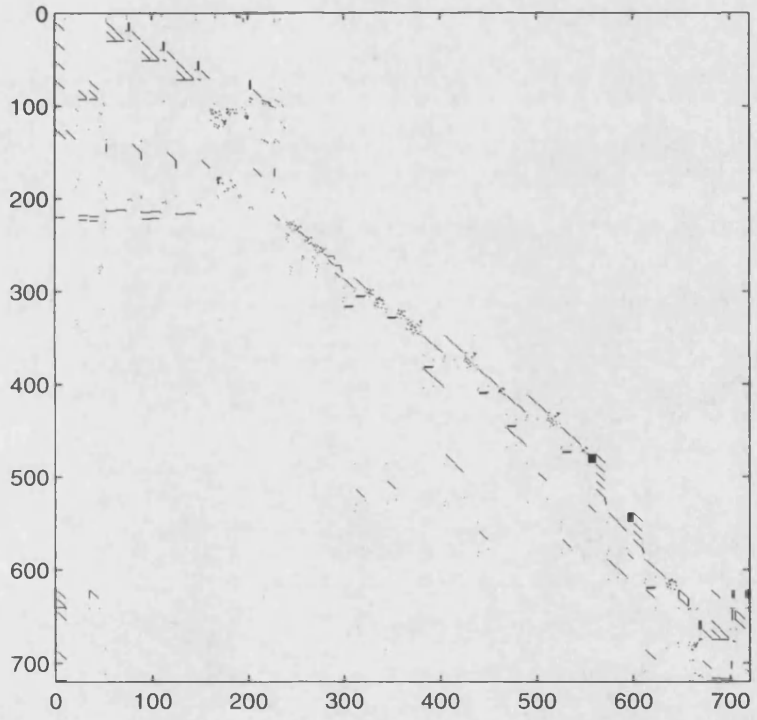

Figure 3-3: *Sparsity pattern of typical Example 2 matrix. The matrix is of size* $n = 719$ *with 2554 non-zero entries.*

3.3.3 Example 3: Plant 1 block 2

This example is from the largest nonlinear block of the model Plant 1. The Jacobian matrices are of size $n = 1279$ with 5553 non-zero entries, and an example sparsity

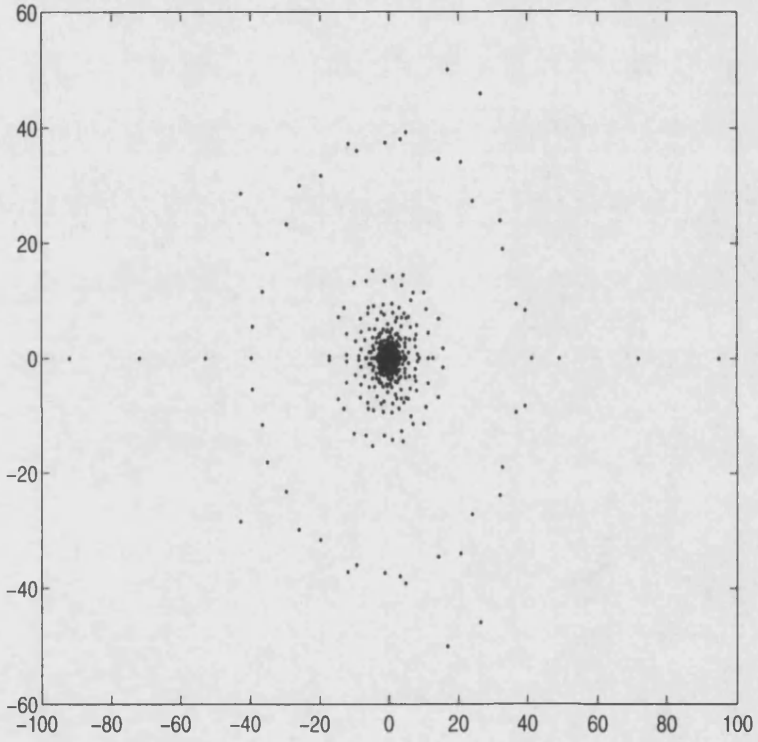

Figure 3-4: *Spectrum of typical Example 2 matrix.*

pattern is shown in Figure 3-5. A typical condition number is $\kappa_2(A) = 3.75 \times 10^{12}$. The spectrum of an example Jacobian is shown in Figure 3-6. The matrices have a density of $\mathcal{D} = 3.4 \times 10^{-3}$.

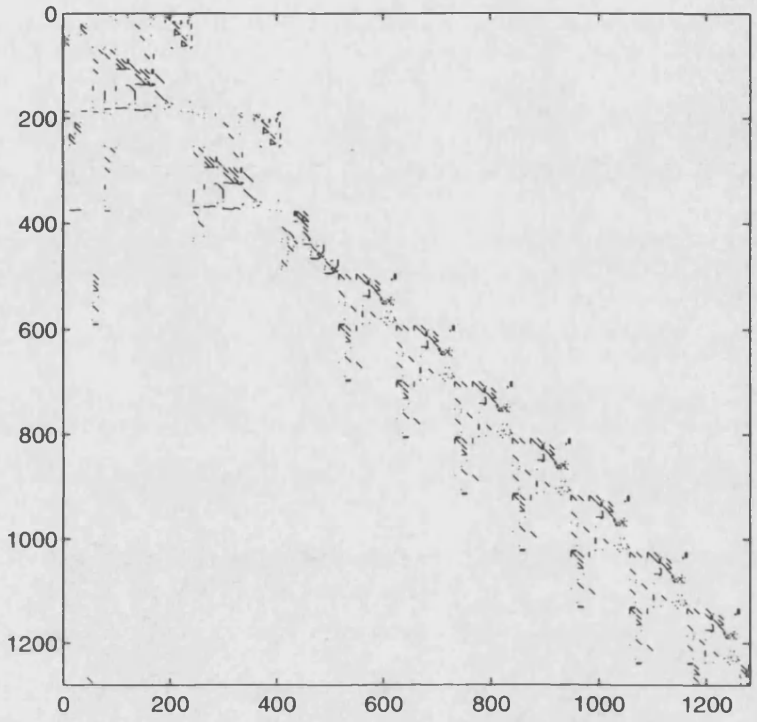

Figure 3-5: *Sparsity pattern of typical Example 3 matrix. The matrix is of size* $n = 1279$ *with 5553 non-zero entries.*

3.3.4 Example 4: Plant 2

This set of matrices arise from another chemical plant model. These Jacobians are of size $n = 3787$ with 16067 non-zero entries, with a a density of $\mathcal{D} = 1.1 \times 10^{-3}$. A typical sparsity pattern is shown in Figure 3-7. The spectrum of an example Jacobian is shown in Figure 3-8, and this has a condition number of 7.34×10^{15}.

Recall from Chapter 1 that although for these matrices, n ranges from approximately 700 to 4000, the number of variables in an entire SPEEDUP flowsheet may be much larger, up to 60,000.

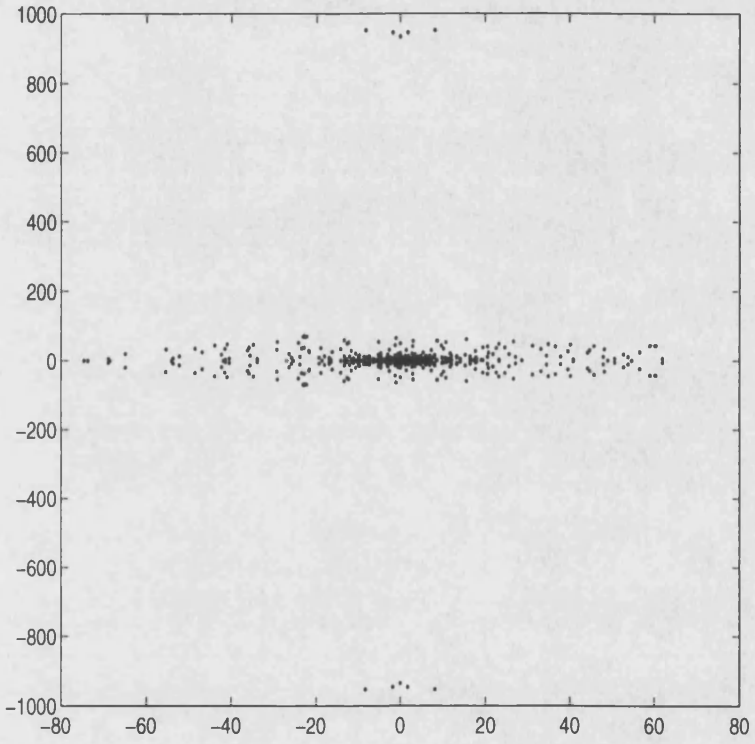

Figure 3-6: *Spectrum of typical Example 3 matrix.*

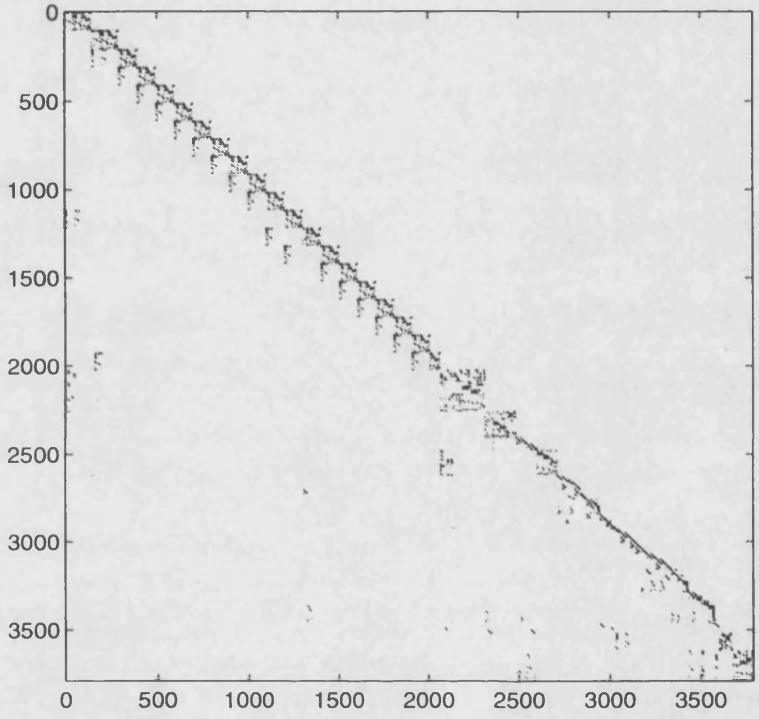

Figure 3-7: *Sparsity pattern of typical Example 4 matrix. The matrix is of size* $n = 3787$ *with 16067 non-zero entries.*

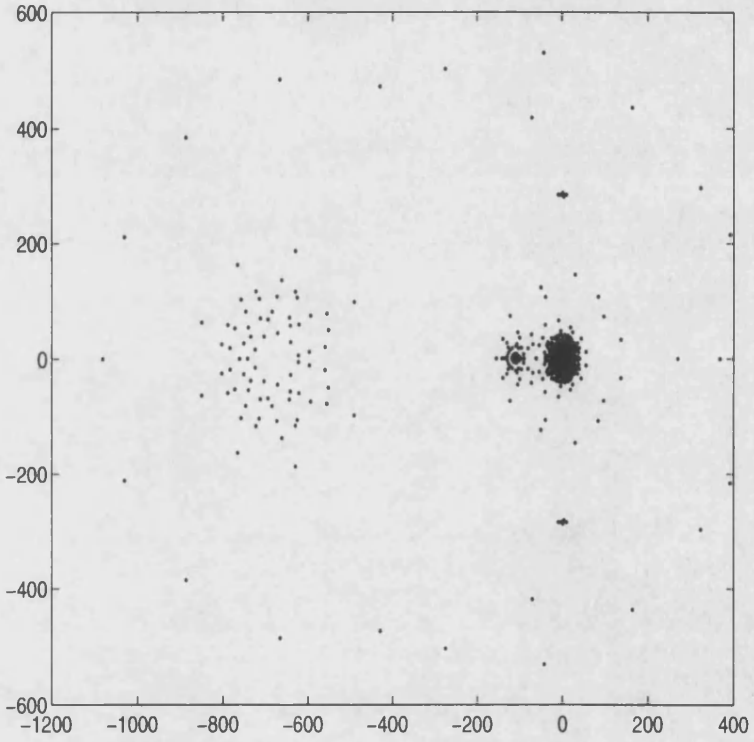

Figure 3-8: *Spectrum of typical Example 4 Jacobian matrix.*

3.4 Incomplete LU factorisation

3.4.1 ILU: An overview

The Incomplete LU factorisation method (ILU) is a popular choice of preconditioner for many applications. Many varieties of this type of preconditioner have been suggested, with early results from Meijerink and van der Vorst [47], Kershaw [39] and Gustafsson [33], with more recent contributions from Axelsson *et al.* [3, 5, 6]. Other references include [11, 24, 48, 50, 51, 69], although this list is by no means exhaustive.

ILU is based on the LU factorisation method. The LU factorisation method [31] for solving linear systems works by factoring the matrix into two matrices, one lower triangular (L) and the other upper triangular (U) such that

$$LU = A,$$

and is equivalent to factorised Gaussian elimination (such methods are sometimes referred to as *direct methods*). To improve the stability of the method, it is usual to employ a pivoting strategy so that growth in the factored elements does not become too large (pivoting methods are discussed further in Chapter 4). Stability of the LU factorisation is discussed by Wilkinson [72], Duff *et al.* [22] and Reid [55], amongst others. This method works well for small, dense matrices, but when applied to large sparse systems, the use of pivoting can cause large amounts of fill-in, resulting in the factored matrices L and U having many times the number of non-zero entries of A. Strategies to counter fill-in can be used (see §4.3), but the calculation of a factorisation that is stable and does not suffer from excess fill-in can be very expensive, and may not be possible.

Instead, we will consider an incomplete factorisation of the form

$$\hat{L}\hat{U} \simeq LU = A, \tag{3.2}$$

where $\hat{L}$ and $\hat{U}$ are constructed to reflect the sparsity of A. This factorisation will be

used as a preconditioner, resulting in the system

$$\hat{U}^{-1}\hat{L}^{-1}Ax = \hat{U}^{-1}\hat{L}^{-1}b \tag{3.3}$$

to which an iterative method (in our case the GMRES algorithm) is applied. Of course, the system $\hat{U}^{-1}\hat{L}^{-1}A$ is not formed explicitly, rather the action of first A, with a matrix-vector product, and then the action of $\hat{L}^{-1}$ and $\hat{U}^{-1}$, carried out with forward and backward solutions, on a vector is calculated to obtain the required result.

If we define the set of index pairs for the non-zero elements of A as

$$\mathcal{S} = \{(i,j) : a_{ij} \neq 0\}, \tag{3.4}$$

then the simplest incomplete factorisation we could have is to insist that

$$p_{i,j} = 0 \ \forall \ (i,j) \notin \mathcal{S}, \tag{3.5}$$

where P represents the sum of the matrices $\hat{L}$ and $\hat{U}$. This approach allows no fill-in to occur in the preconditioner, and so has the advantage of using a fixed amount of storage and arithmetic. This variant is sometimes referred to as ILU(0), indicating explicitly zero fill-in. The simplest implementations of ILU factorisations do not use any kind of pivoting strategy, and are generally used for strictly diagonally dominant matrices such as those that arise from certain discretised PDE problems.

ILU is essentially a black-box solver - it makes no use of the structure or other special properties of a matrix to which it is applied. Whilst this is a disadvantage for problems where such structure or properties exist, this is not a drawback for matrices arising from SPEEDUP problems, which do not possess any properties that can be utilised. This means that we are able to use an off-the-shelf package to study the effectiveness of this type of preconditioner.

3.4.2 Modified ILU

Many variations of the ILU method defined by (3.5) are possible. In some variants, entries that would have been discarded for not being in the set $\mathcal{S}$ are instead added to the corresponding diagonal entry. This approach, called modified ILU (MILU), was proposed by Gustafsson [33] and Axelsson and Munksgaard [7]. It has the advantage over ILU that whilst using the same amount of storage as ILU, it preserves more of the properties of the matrix, such as row sums. For example, this method is generally found to produce better results for matrices arising from finite element methods than the standard ILU method.

Another way of modifying the factorisation is to allow a certain degree of fill-in to occur. This can be done by either allowing fill-in in certain areas of the factorisation (fill-in *by position*), or by discarding elements that are smaller than some specified tolerance (fill-in *by value*). Such a tolerance is called a *drop tolerance*, see Axelsson [3, §7.1]. By varying the size of the drop tolerance, the amount of fill-in can be controlled. This approach has the disadvantage that the amount of storage needed for the factorisation cannot be determined *a priori*.

A third approach is to set a *level of fill*, `lfil`, and only allow the largest `lfil` entries occurring in each row to add to the fill-in. In the numerical experiments presented in §3.4.4, we will use several varieties of ILU preconditioners, employing both drop tolerances and fill-level strategies, as well as the more basic ILU methods.

3.4.3 Stability and Existence

Two issues of importance for such factorisations are existence and stability: We need to know that for a given matrix A a factorisation exists, and if one does, how meaningful in the context of finite precision arithmetic it will be. Stability of incomplete factorisations is considered in the same manner as that of complete factorisations, by considering a 'growth factor' ([4, 22]), defined as

$$\max_{i,j,k} |a_{i,j}^{(k)}| / \max_{i,j} |a_{i,j}| \geq 1,$$

where $a_{i,j}^{(k)}$ is the ijth entry in the partially factorised matrix $A^{(k)}$ occurring at the kth step of the factorisation. A factorisation (incomplete or not) is said to be stable if this growth factor remains reasonably bounded.

Results regarding stability of incomplete factorisations have been proved for classes of matrices such as those arising from discretised PDE's (Elman, [27]), and other classes of matrices, described in Axelsson and Barker [4]. Meijerink and van der Vorst [47] give results for stability for M-matrices. Existence results for factorisations of various classes of matrices have been discussed by Axelsson and Barker [4], Axelsson and Lindskog [5] and Buoni [11] amongst others.

For a general nonsymmetric matrix such as a SPEEDUP Jacobian, there are no results in the literature regarding stability or existence of ILU factorisations. Indeed, the ill-conditioning of the matrices suggests that problems may occur in the calculation of such factorisations. In the following section, we present results of experiments conducted with ILU preconditioned GMRES on the examples outlined in section 3.3.

3.4.4 Numerical experiments

In this section we present results of using various types of ILU preconditioner on the examples outlined in the previous section. The preconditioners and GMRES solver used are from Saad's SPARSKIT library of routines [60]. Four variants of ILU are used: ILU(0), MILU(0) and two variants with a dual strategy for allowing fill-in, ILUT and ILUTP. The dual strategy employed by ILUT consists of specifying two parameters, a drop tolerance `tol` and a value `lfil`. Any element with modulus less than `tol` relative to the norm of the current row in U is discarded, and only the `lfil`$+l_i$ elements are kept in the ith row of L and the `lfil`$+u_i$ elements in the ith row of U, where l_i and u_i are the number of elements in row i of L and U respectively. ILUTP uses the above strategies in conjunction with partial pivoting to attempt to improve the stability of the factorisation. The methods that do not feature pivoting require that the diagonal of the matrix be free of zeros. This feature is not guaranteed for general SPEEDUP Jacobians, so some pre-processing was required. The Harwell routine MC21A computes a permutation of a matrix to provide a non-zero diagonal, details of which can be

found in [20]. This algorithm was used to ensure the Jacobians satisfied this criterion. The GMRES solver is implemented with a selective reorthogonalisation strategy in the modified Gramm-Schmidt process. Unlike the Brown and Hindmarsh strategy (§2.7.2), extra vector products are used to check orthogonality of the new basis vector against all previous ones, and reorthogonalisation is performed if required.

Example 1

Figures 3-9, 3-10 and 3-11 show plots of the log of the norm of the GMRES residual against iteration number for these various preconditioners applied to a sample BTX Jacobian matrix. For the examples using the restarted GMRES algorithm 2.3 shown in Figure 3-11, the number of inner iterations is shown rather than the number of outer (GMRES(m)) iterations. GMRES was limited to a maximum of 200 (inner) iterations. For application to real-time solution methods on realistic problems, we require convergence to occur in much fewer iterations than this. Data for MILU(0) preconditioning are

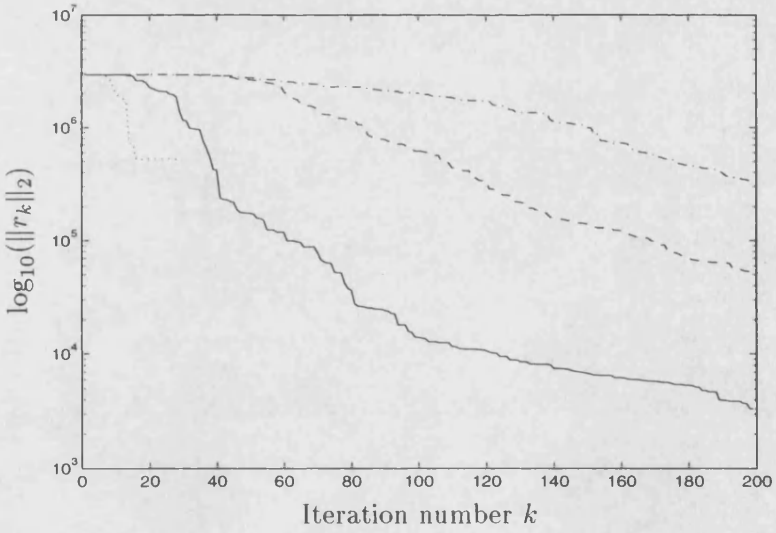

Figure 3-9: *Residual plots for ILU(0) (dotted line) and ILUT preconditioned GMRES with* `lfil`*=1 (solid line), 7 (dashed line) and 12 (dot-dashed line) applied to a problem from Example 1.* `tol` *was set to* 1×10^{-4}.

not shown on any of the figures, since the SPARSKIT routine was unable to compute a factorisation due to encountering zero entries on the diagonal. GMRES preconditioned

with ILU(0) breaks down after 28 iterations. Saad [61] showed that GMRES cannot break down when applied to a non-singular system: If we recall from Algorithm 2.2 step 2, the new basis vector v_{k+1} is given by

$$v_{j+1} = \hat{v}_{j+1}/h_{j+1,j}.$$

Thus v_{k+1} can be formed provided that $h_{j+1,j} \neq 0$. If, however, $h_{j+1,j} = 0$ at some step in the iteration, then a breakdown is possible. Such a breakdown is called a *happy breakdown*, and the current iterate x_j is the exact solution. Thus GMRES cannot break down without producing the exact solution. The ILU(0) breakdown must therefore be due to the preconditioned system being singular or near-singular, most likely resulting from numerical instability of the incomplete factorisation.

The performance of the ILUT preconditioner decreases as the level of fill-in is allowed to increase. This may be explained by the lack of pivoting. If pivoting is not performed, then the factorisation will not be stable, since small (nearly zero) pivots will be used. The more entries that are allowed in the factorisation, the larger the growth factor due to these small pivots will be, thus resulting in a more inaccurate preconditioner. The plots for ILUTP, which employs partial pivoting, show a marked increase in performance as the level of fill-in is allowed to increase. However, the performance is still very slow, requiring many GMRES iterations to reduce the residual to a reasonable level. These results were calculated using full GMRES, which is too expensive for practical purposes, especially in time-critical applications such as SPEEDUP. The preconditioned systems are still ill-conditioned, producing inaccurate residual norm estimates (see §§2.3.4, 2.7); for example, the ILUTP preconditioned GMRES terminates with an estimated residual norm of 2.88×10^{-5}. However, the actual preconditioned residual norm is 2.61×10^{-2} and the error norm is 1.24×10^{-2}. Thus whilst GMRES appears to be converging, the results are not as accurate as is suggested by the approximate residual norms.

Figure 3-11 shows residual norms for ILUTP preconditioned GMRES(m) with `lfil`=12 for several values of m. As can be seen, the iteration stalls for m=10 and m=50 once GMRES is restarted, and no progress is made in reducing the residual norm until much later in the second outer GMRES(100) iteration. It appears from the results presented above that using ILU-type preconditioners on matrices from Example 1 produces poor

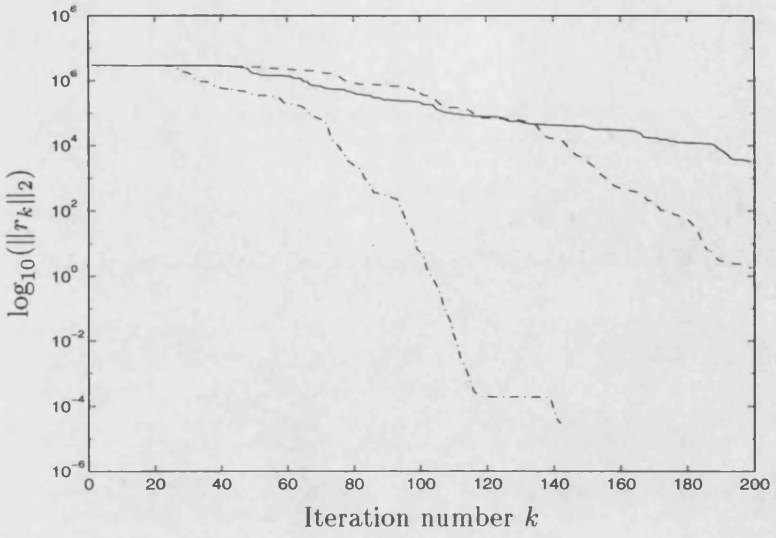

Figure 3-10: *Residual plots for ILUTP preconditioned GMRES applied to a problem from Example 1, with* `lfil`*=1 (solid line), 7 (dashed line) and 12 (dot-dashed line).* `tol` *was set to* 1×10^{-4}.

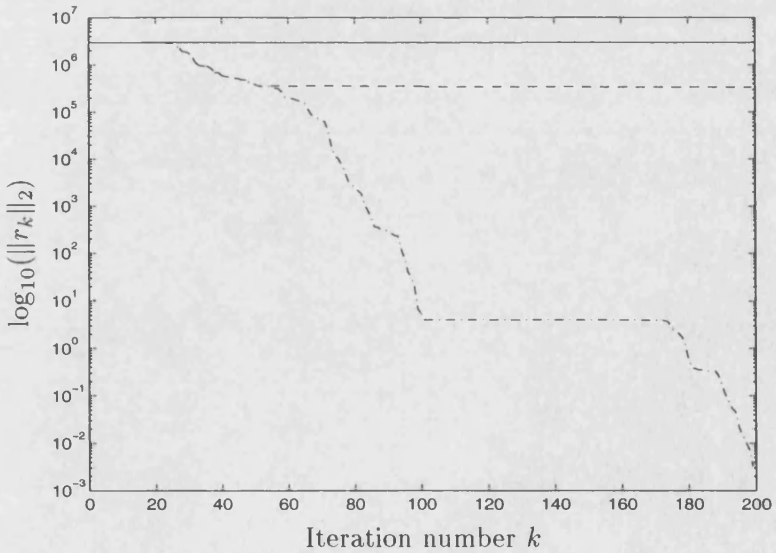

Figure 3-11: *Residual plots for ILUTP preconditioned GMRES(m) applied to a problem from Example 1, with* `lfil`*=12, m=10 (solid line), m=50 (dashed line) and m=100 (dot-dashed line).* `tol` *was set to* 1×10^{-4}.

and inaccurate convergence of the GMRES algorithm. The poor convergence can be attributed to an unfavourable distribution of the spectrum of the preconditioned system, and the inaccuracy may be caused by a loss of orthogonality in the basis vectors for $\mathcal{K}_k$. The spectrum of the preconditioned system is considered in §3.4.5, but first we will consider the inaccuracy of the norm estimates.

As was outlined in §2.7, inaccuracy in the orthogonalisation process leads to inaccurate residual norm estimates. The GMRES solver used for our examples uses a reorthogonalisation procedure to attempt to reduce the effect of this. We can gauge how much the vectors are suffering from loss of orthogonality by examining how often reorthogonalisation is required. For the example presented above, many reorthogonalisations were required on every GMRES iteration, implying a massive loss of orthogonality in the subspace vectors. This is probably caused by the preconditioned system being nearly numerically singular; whilst in exact arithmetic the MGS process would produce orthogonal vectors, the poor numerical properties of the matrix, and hence the preconditioner, mean that errors caused by roundoff can cause the preconditioned matrix to act as if it was singular, thus producing vectors with a high degree of linear dependence.

Example 2

If we apply the same preconditioners to our second example, we obtain apparently similar results, but the preconditioned systems appear to have worse conditioning than those for Example 1. The ILUT preconditioner with `lfil`=1 and `tol`=1×10^{-4} yields an estimated GMRES residual norm of 1.256 after 200 iterations, but the actual residual norm is 1.57×10^{16}. The least inaccurate results are obtained from the ILUTP preconditioner with `lfil`=12 and `tol`=1×10^{-4}, with an estimated residual norm of 2.09×10^{-8} after 78 iterations, with an actual residual norm of 5.23×10^{-3} and an error norm of 1.426. Plots of the estimated residual norms for GMRES preconditioned with ILU(0), ILUT with `lfil`=7 and ILUTP with `lfil`=7 and 12 can be seen in Figure 3-12. `tol` was set to 1×10^{-4}. The MILU method again failed to create a factorisation due to encountering zeros on the diagonal.

We can see that the poor performance of ILU preconditioned GMRES experienced with

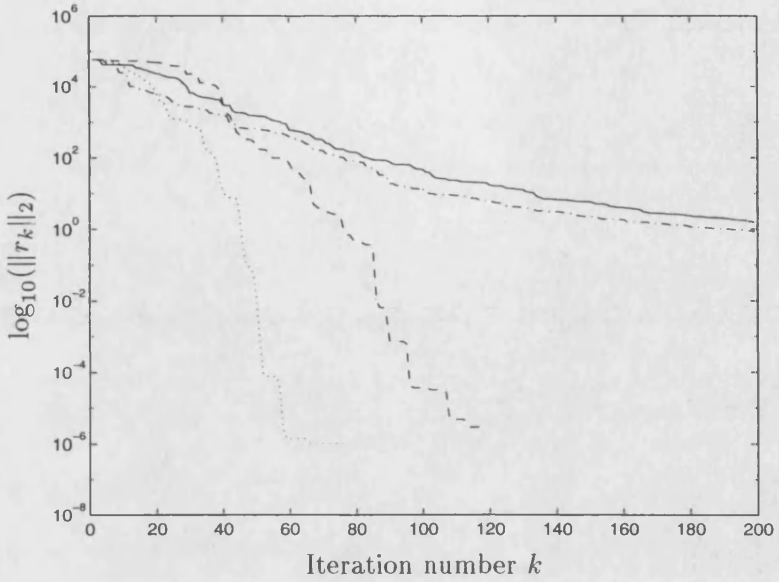

Figure 3-12: *Residual norm estimates for GMRES preconditioned with ILU(0) (solid line), ILUT with* `lfil`*=7 (dot-dashed line) and ILUTP with* `lfil`*=7 (dashed line) and 12 (dotted line) applied to a matrix from Example 2.* `tol`$=1 \times 10^{-4}$.

the first example is repeated here. If we examine how the non-orthogonality check is behaving, then we find that as before, many reorthogonalisations are required on each GMRES iteration, again suggestive of near numerical singularity in the preconditioned matrix. The inaccuracy and slow convergence shown by these two examples mean that ILU preconditioning will not meet the requirements needed for a SPEEDUP iterative linear solver.

3.4.5 Approximating $\sigma(A)$

These first two examples show that ILU preconditioning and its variants, whilst demonstrated by the literature to be useful preconditioners for matrices that arise from problems such as the application of the finite element method to the solution of PDE's, are not effective as preconditioners for matrices arising from SPEEDUP. Since the preconditioned matrix $M^{-1}A$ is not formed explicitly, we cannot calculate its spectrum $\sigma(M^{-1}A)$ to use results from Chapter 2 to explain the poor performance. However, we are able to gain an approximation to it by using the fact that GMRES is based on Arnoldi's method. This method approximates the spectrum of a matrix A with that of the upper

Hessenberg matrix H_k from algorithm 2.1. Thus we can approximate $\sigma(M^{-1}A)$ with the spectrum of the upper Hessenberg matrix $\sigma(H_k)$ from the GMRES algorithm. Using this will enable us to make an estimate on the distribution of the spectrum of the preconditioned matrix to show how the preconditioning has affected the distribution of $\sigma(A)$. We will approximate the spectrum of $M^{-1}A$ by that of H_k where k is the number of GMRES iterations required to reduce the initial (preconditioned) residual norm by a factor[1] of 1×10^{-11}.

Figure 3-13 shows the spectrum of H_{150} from GMRES applied to a matrix from Example 1, preconditioned with ILUTP with `lfil`=12 and `tol`=1×10^{-4}. The plot on the left shows one extremal eigenvalue at $\lambda \approx -5000$, and the plot on the right shows the distribution of the eigenvalues that are situated closer to the origin. Comparing this spectrum with the unpreconditioned spectrum shown in Figure 3-2, then the distribution has been improved significantly by the preconditioner, but due to the distribution of the eigenvalues close to the origin, it can be seen that we are unable to effectively apply any results from Chapter 2. However, numerical results presented in §3.4.6 show that even in the absence of such theoretical results, we can still demonstrate that convergence will be slow.

If we examine a matrix from Example 2, then we find similar results: Taking the matrix H_{68} from GMRES preconditioned with ILUTP, we find the condition number $\kappa_2(H_{68}) = 5.16 \times 10^{10}$. This is not as large as that from Example 1; this may be explained by the fact that we are using a matrix that has been generated from fewer GMRES iterations. Results have indicated that for these examples, $\kappa_2(H_k)$ increases with k. Calculating the spectrum of H_{68} (Figure 3-14) shows a similar degree of improvement in the conditioning as was found in Example 1; the eigenvalues are not as widely spread as those of the original matrix, but the distribution still indicates that GMRES convergence will be slow.

We will now present the idea of a matrix pseudospectra, and use this to perform some analysis of the incomplete factorisations to help explain their poor performance.

[1] Based on the GMRES residual norm estimate, not the true residual norm

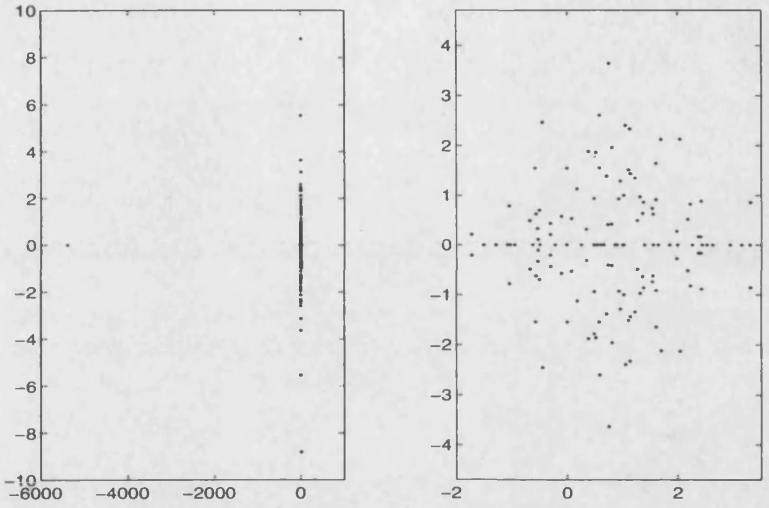

Figure 3-13: *Spectrum of H_{150} from ILUTP preconditioned GMRES applied to a matrix from Example 1. The left-hand pane shows the whole spectrum, whilst the right-hand pane is a close-up of the main cluster situated around the origin.*

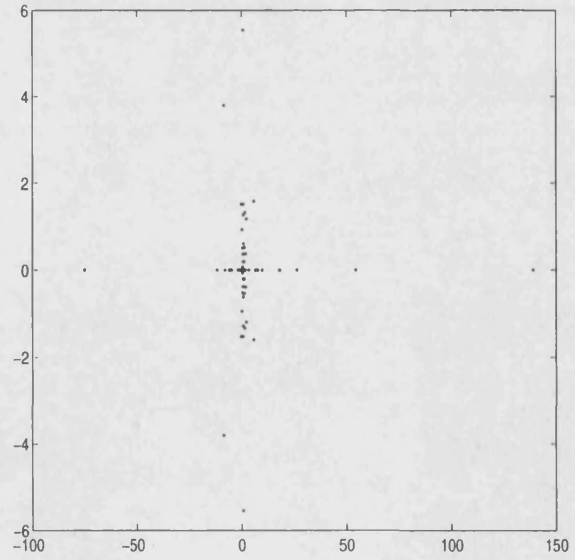

Figure 3-14: *Spectrum of matrix H_{68} from GMRES algorithm preconditioned with ILUTP, applied to a matrix from Example 2.*

3.4.6 Pseudospectra

The $\epsilon-$*pseudospectrum* of a matrix A can be defined for an $\epsilon \geq 0$ as

$$\sigma_\epsilon(A) = \{z \in \mathbb{C} : \ \|(zI - A)^{-1}\| \geq \epsilon^{-1}\}. \tag{3.6}$$

Thus the $\epsilon-$pseudospectra of A are nested closed sets with $\sigma_0(A) = \sigma(A)$. An alternative definition to (3.6) is given by

$$\sigma_\epsilon(A) = \{z \in \mathbb{C} : \ z \in \sigma(A + E), \|E\| \leq \epsilon\}. \tag{3.7}$$

Thus the $\epsilon-$pseudospectrum of A can be viewed as the set of eigenvalues of the set of all matrices perturbed from A by an amount ϵ. In [66], Trefethen notes that if a matrix is normal, then the pseudospectra of A are merely the $\epsilon-$neighborhoods around the spectrum of A. For non-normal matrices, this is not the case. If a matrix is only slightly non-normal, i.e. $\kappa_2(U)$ is not much greater than unity, then we can expect the pseudospectra to be close to the $\epsilon-$neighborhoods around the spectrum of A. If we recall from Chapter 2 the example matrix A_ρ, then we can plot the pseudospectra for various values of ϵ. Figure 3-15 shows the level curves for $\epsilon = 0.3, 0.2, 0.1, 0.05, 0.03$ and 0.01. For these values, the level curves are situated close to the eigenvalues of A, thus indicating that A is not far from normal ($\|A_\rho^T A_\rho - A_\rho A_\rho^T\| = 1.79 \times 10^{-4}$).

We can make two remarks with regard to how the normality of a matrix effects the convergence of GMRES:

- Recalling Theorem 2.3, a large value of $\kappa_2(A)$ will give a pessimistic bound for the rate of convergence, regardless of the distribution of $\sigma(M^{-1}A)$.
- In their conclusions in [49], Nachtigal *et al.* note that if a matrix A is far from normal, then convergence is slower by a potentially unbounded factor than the distribution of the eigenvalues alone would suggest, and that convergence is approximately determined by the pseudospectra of A.

These remarks give an insight into how we can use pseudospectra to examine the conver-

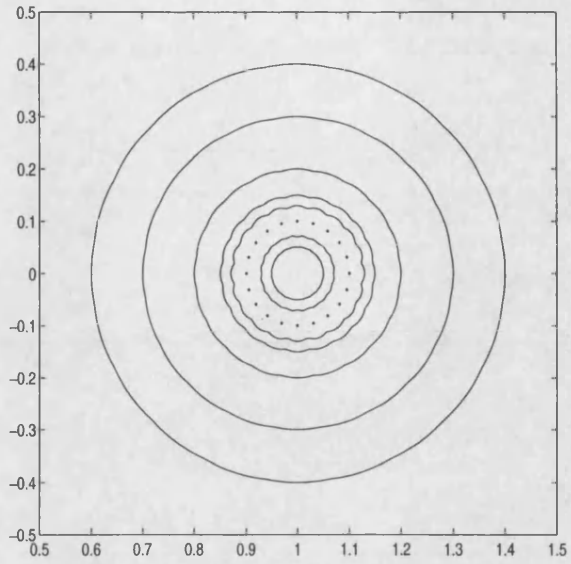

Figure 3-15: *Plot of pseudospectra of example* A_ρ, *with* $\epsilon = 0.3, 0.2, 0.1, 0.05, 0.03$ *and* 0.01.

gence behaviour of GMRES. Plots of the pseudospectra of the preconditioned matrices will show how far from normal they are, and if a large degree of non-normality is detected, then from the two remarks above we should expect poor convergence of GMRES.

As has already been mentioned, we cannot explicitly form the matrix $M^{-1}A$, since in general we only have the action of the preconditioner M^{-1} on a vector. This means that we are unable to calculate the pseudospectra directly. In fact, even if we could form $M^{-1}A$, for the example matrices presented, the size of the systems means such a calculation would be prohibitively expensive. Toh and Trefethen [65] suggest that instead of approximating the spectrum of A with that of H_k, one approximates the pseudospectrum [66] of A with that of H_k. In [65], the number of Arnoldi iterations required to yield an accurate approximation to the pseudospectra of various examples is discussed. For our purposes, we will use the matrix H_k where k is the number of GMRES iterations required to reduce the initial residual norm by a factor[2] of 1×10^{-11}.

Figure 3-16 shows the pseudospectra of the upper Hessenberg matrix H_{150} from ILUTP preconditioned GMRES applied to a matrix from Example 1, for $\epsilon = 1 \times 10^{-2}$, 5×10^{-3}, 1×10^{-3} and 5×10^{-4}. The large outlying eigenvalue of H_{150} at $\lambda \approx -5000$ has

[2]Based on the GMRES residual norm estimate, not the true residual norm.

been omitted from this figure. In contrast to figure 3-15, this figure shows that for small values of ϵ, the matrix is very sensitive to perturbation. This suggests a large degree of non-normality, and calculating $\kappa_2(U)$ gives a value of 1.63×10^{41}, where U is the matrix formed from the eigenvectors of H_{150}.

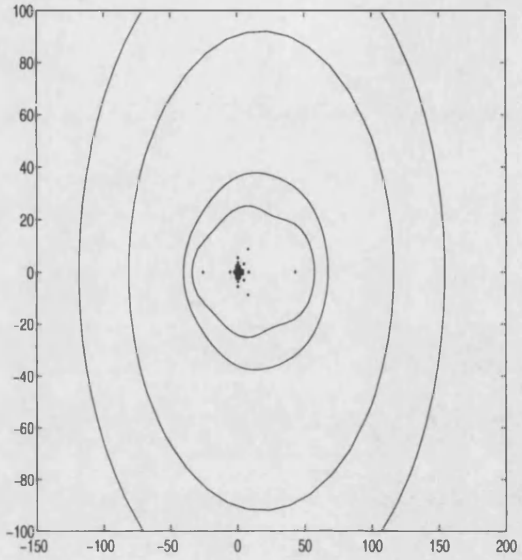

Figure 3-16: *Plot showing a portion of the pseudospecta of the matrix H_{150} from GMRES applied to a matrix from Example 1, preconditioned with ILUTP with* `lfil=12` *and* `tol`$= 1 \times 10^{-4}$. *The level curves are for* $\epsilon = 1 \times 10^{-2}, 5 \times 10^{-3}, 1 \times 10^{-3}$ and 5×10^{-4}.

These two remarks are justified both by our experimental results, showing the poor convergence of GMRES, and by the plots of the pseudospecta of H_{150} which show the large degree of non-normality in the upper Hessenberg matrix. Toh and Trefethen [65] suggest that for a sufficiently large value of k, the pseudospectra of H_k is an accurate approximation to that of the matrix A to which the Arnoldi method is applied. By choosing k large enough such that GMRES has 'converged', we hope that we have obtained an accurate approximation to the pseudospecta of our example matrix. If this is so, then we can expect the same large degree of non-normality demonstrated by H_{150} to be exhibited by the preconditioned matrix, thus reinforcing the claim that convergence will be slow for this example.

Calculating the pseudospectra of H_{68} from Example 2 yields similar looking plots (Figure 3-17) to those from Example 1. We find that the plots of the level curves indicate

a large degree of non-normality, also implying slow convergence of GMRES.

These two examples show that ILU-type preconditioning is not an effective method for accelerating the convergence of GMRES applied to SPEEDUP Jacobian matrices.

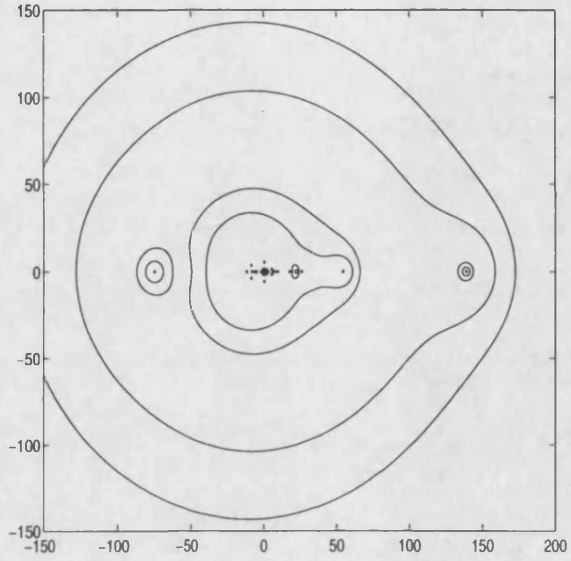

Figure 3-17: *Pseudospectra of matrix H_{68} from ILUTP preconditioned GMRES applied to a matrix from Example 2. The level curves are for $\epsilon = 1 \times 10^{-2}$, 5×10^{-3}, 1×10^{-3} and 5×10^{-4}.*

One feature of a preconditioner that would be attractive for application in SPEEDUP problems would be reusability. We are concerned with the solution of a sequence of linear systems (1.1), so once a preconditioner M_1^{-1} has been calculated for an initial matrix J_1, can it be used as a preconditioner for subsequent matrices J_l, $l = 2, \ldots$? From the results presented above for the ILU variants, this is unlikely to be a possibility since ILU is not an effective enough preconditioner for J_1. This leads us to seek a preconditioning method which can satisfy this reusability criterion.

3.5 Full LU preconditioning

3.5.1 Motivation

Due to the poor performance of the ILU preconditioners, we are forced to seek an alternative approach to precondition the matrices J_l. In §3.4.1, fill-in was allowed on some subset of the indices $\{(i,j), i,j = 1,\ldots n\}$, and so if we take that set equal to $\{(i,j), i,j = 1,\ldots n\}$ we will calculate a full factorisation of J_l,

$$\hat{L}\hat{U} = LU = J_l \tag{3.8}$$

(cf. (3.2)). Clearly, this calculation negates the need for an iterative solution of the system

$$J_l x = b_l. \tag{3.9}$$

However, since we have a series of systems (1.1) to solve, then we could use a factorisation (3.8) of the initial matrix J_1 to precondition the subsequent Jacobians. We can see how this approach could be effective by considering the quantity

$$\|I - M^{-1}J_l\|.$$

If this quantity is 'small' in some sense, then our preconditioned system is 'close' to the identity, and we can expect reasonably fast convergence of GMRES. If we consider the sequence of matrices we will be solving (1.1) in the form

$$J_l = J_1 + D_l,\ l = 2,\ldots,$$

then if we take $M_1 = J_1$, this quantity can be re-expressed as

$$\|M_1^{-1}D_l\|.$$

So if $\|M_1^{-1}D_l\|$ is small, then the preconditioned system is close to the identity in some sense, and so should have eigenvalues clustered around 1. If the clustering is reasonably tight, then we can expect rapid convergence of GMRES from Theorem 2.7.

We can reasonably expect $\|M_1^{-1}D_l\|$ to be small if $\|D_l\|$ is small, so if the sequence of differences D_l are consistently small, then we may be able to use one preconditioner for many Jacobians J_l.

3.5.2 Numerical experiments: Example 3

Results of such a preconditioner applied to some Jacobian matrices from Example 3 are shown in Table 3.1. This shows the number of GMRES iterations required to reduce the residual norm estimate to satisfy

$$\|r_k\|_2 \leq 1.0 \times 10^{-8}. \tag{3.10}$$

The values of $\|D_l\|$ were calculated using MATLAB's **normest** function, which is based on the power method. These results show that convergence is possible with much smaller values of k than for ILU preconditioning, in spite of the large values of $\|D_l\|$.

l	k	$\|D_l\|$	$\|e_k\|_2$
2	3	1.64×10^2	2.61×10^{-6}
3	17	3.37×10^2	1.64×10^{-6}
4	7	1.64×10^2	1.50×10^{-4}
5	36	1.32×10^3	8.55×10^{-7}
6	3	1.64×10^2	5.55×10^{-3}

Table 3.1: *Results of using $M_1 = J_1$ as a preconditioner for GMRES applied to Jacobian matrices from Example 3. Table shows the value of k required for GMRES residual norm estimate to be less than 1×10^{-8}, the value of $\|D_l\|$ and the value of $\|e_k\|_2$.*

Another advantage we find with this type of preconditioning is improved accuracy: Whereas with the ILU preconditioning method, the GMRES algorithm was producing inaccurate residual norm estimates, using an exact inverse-type preconditioner gives good agreement between estimated and true residual norms. For example, when preconditioning matrix J_5 with the exact inverse of J_1, GMRES produces a residual norm estimate of 3.230×10^{-9} after 36 iterations, whereas the true residual norm $\|r_{36}\|_2 = 3.234 \times 10^{-9}$. This method also yields a reasonably good error norm of $\|e_{36}\|_2 = 8.550 \times 10^{-7}$. The worst case is that of J_6, which yields a residual norm

of 3.82×10^{-9} but an error norm of 5.55×10^{-3}. This could be caused by the residual containing large components in eigendirections corresponding to the two small eigenvalues $\lambda_1 = 3.40 \times 10^{-2}$ and $\lambda_2 = 3.48 \times 10^{-2}$ of the preconditioned matrix. If GMRES is restarted with a tighter tolerance of TOL$= 1 \times 10^{-10}$, then an additional four iterations produces an approximate solution with an improved error of 1.98×10^{-9}.

Of all the Jacobians we consider in this example, the matrix J_5 requires the largest number of GMRES iterations in order to satisfy the convergence criterion (3.10), when preconditioned by the initial Jacobian matrix. Due to this, we might expect difficulty in applying results such as Theorem 2.7. However, this is not the case. Since we are able to explicitly form the preconditioned matrix, we can calculate its spectrum using MATLAB.

3.5.3 Examining the spectrum of the preconditioned system

If we examine the spectrum, we see that all of the eigenvalues are situated in the right half-plane, with a large, well-clustered group situated near the origin, and only a few small eigenvalues to the left of this cluster. This allows us to calculate values for ρ and c that result in a disc enclosing the majority of the spectrum of $M_1^{-1}J_5$. By choosing values of $\rho = 0.47$ and $c = 0.85$, we can enclose 1245 of the 1279 eigenvalues of the matrix, excluding 12 to the left of c and 22 to the right. This is shown in Figure 3-18, which shows the spectrum of $M_1^{-1}J_5$ (omitting the large outliers $\lambda_{1277} \approx 11.3$, $\lambda_{1278} \approx 317$ and $\lambda_{1279} \approx 347$) and the disc centred on 0.85 of radius 0.47. We are unable to calculate a value for ξ (or $\kappa_2(U)$) that are required to apply Theorem 2.7, so we can only apply Remark 2.2 to determine the rate of convergence. From this, we get the result that

$$\|e_k\|_2 = O\left(\frac{0.47}{0.85}\right)^k \approx 0.55^k. \tag{3.11}$$

If we calculate the exact solution, then we can obtain the error at each iteration, and see how well the rate of convergence agrees with the result (3.11). Figure 3-19 shows the error at the kth step calculated using the exact solution, and a bound obtained by multiplying $(\frac{0.47}{0.85})^k$ by a constant. This shows good agreement between the predicted and observed behaviour of the convergence.

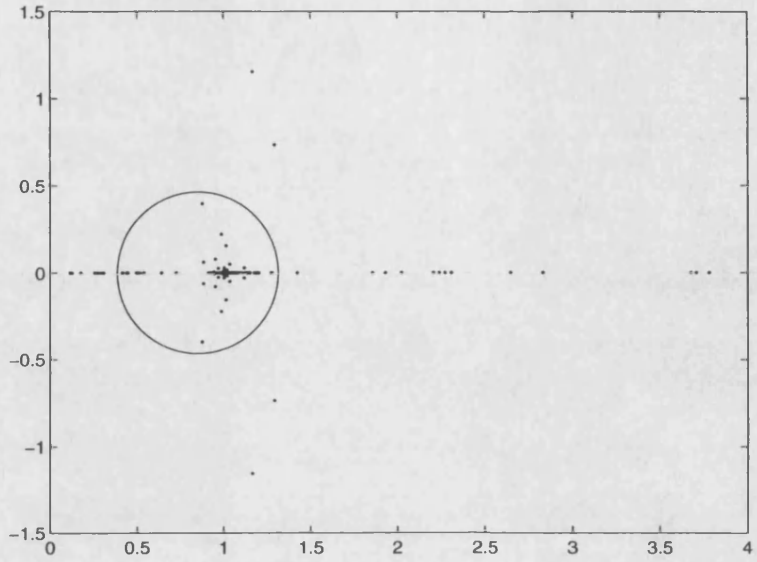

Figure 3-18: *Plot of* $\sigma(M_1^{-1}J_5)$ *(excluding the three largest eigenvalues) and the disc centred on* $c = 0.85$ *of radius* $\rho = 0.47$ *that encloses most of the spectrum.*

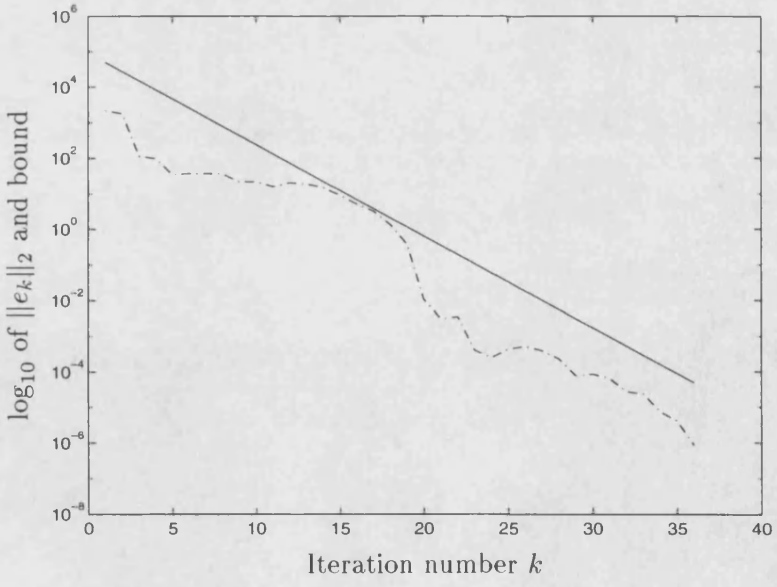

Figure 3-19: *Plots of* $\|e_k\|_2$ *(dot-dashed line) and a bound obtained from (3.11) (solid line) for GMRES applied to* J_5 *from Example 3, preconditioned with an exact inverse of* J_1.

3.5.4 Comparing $\sigma(M^{-1}J_l)$ and $\sigma(H_k)$

As with examples with the ILU preconditioners, we can examine the spectrum of the upper Hessenberg matrix H_k from the GMRES algorithm. This time, we take the value of k to be the number of iterations required to satisfy (3.10). From Table 3.1, k=36 for this example.

The bottom plot in Figure 3-20 shows how the large cluster of eigenvalues of $M_1^{-1}J_5$ around 1 are approximated by only a few eigenvalues of H_{36}. The fact that GMRES has met the convergence criterion (3.10) tells us that the cluster of eigenvalues of $M_1^{-1}J_5$ has been approximated accurately enough by those few eigenvalues of H_{36}. Each GMRES iteration increases the dimension of H_k by one, thus introducing a new eigenvalue; the fact that the cluster was approximated by only a few eigenvalues of H_{36} indicates that not many of the 36 GMRES iterations were required to resolve it to a sufficient accuracy to allow convergence to occur.

The large outlying eigenvalues of $M_1^{-1}J_5$ not shown in Figure 3-20 are approximated well by those of H_{36}. Table 3.2 shows the values of these eigenvalues and their corresponding approximations.

i	λ_i of $M_1^{-1}J_5$	j	λ_j of H_{36}
1277	11.29397	34	11.29397
1278	317.67574	35	317.67574
1279	347.15863	36	347.15863

Table 3.2: *Large eigenvalues of* $M_1^{-1}J_5$ *and* H_{36}.

This example shows that this type of preconditioner yields a system with a spectrum distributed in such a way to allow GMRES to converge faster and with more accuracy than was possible with ILU-type preconditioners. Most of the eigenvalues are clustered tightly around 1, and thus require few GMRES iterations to resolve them.

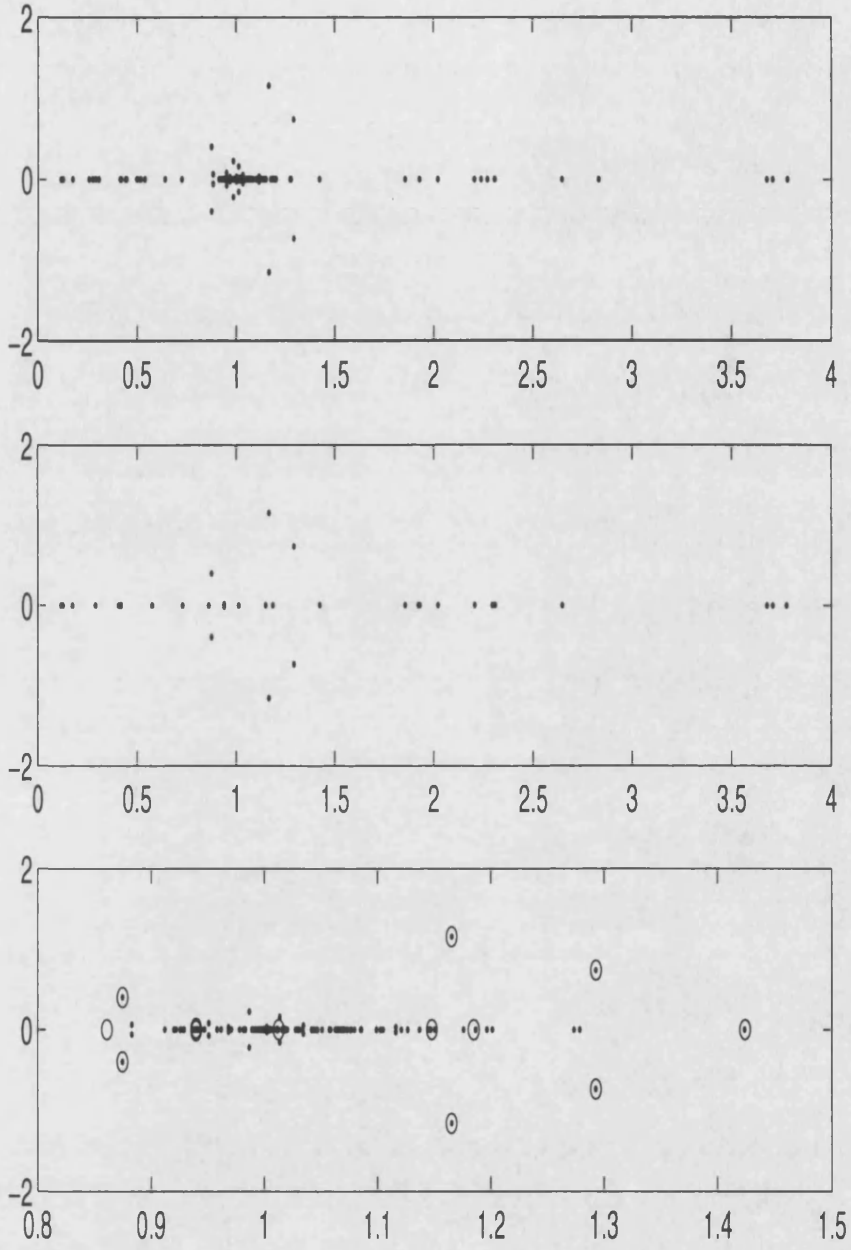

Figure 3-20: *Top: Spectrum of $M_1^{-1}J_5$, excluding large outliers. Middle: Spectrum of H_{36} from GMRES applied to $M_1^{-1}J_5$, excluding large outliers. Bottom: Eigenvalues of $M_1^{-1}J_5$ (dots) and H_{36} (circles) situated near 1.*

3.5.5 Pseudospectra of H_k from full LU preconditioned GMRES

Finally, we will examine the pseudospectra of the matrix H_{36} as we did for the ILU preconditioning methods. For the ILU examples, we found the pseudospectra of the matrices H_k from ILU-preconditioned GMRES indicated a large degree of non-normality and sensitivity to perturbation in the matrix H_k, behaviour which Toh and Trefethen [65] suggest will be reflected in the original (preconditioned) matrix.

Figure 3-21 shows the pseudospectra of H_{36} from GMRES applied to $M_1^{-1}J_5$, with values of $\epsilon = 1 \times 10^{-2}$, 5×10^{-3}, 1×10^{-3} and 5×10^{-4}. If we compare this figure to either of Figures 3-16 or 3-17, we can see that the level curves for corresponding values of ϵ are situated a lot closer to the spectrum of H_k, indicated that the matrix H_{36} is substantially less non-normal than H_{150} (Example 1) and H_{68} (Example 2) generated by ILU-preconditioned GMRES. If we calculate the eigenvectors of H_{36}, we find that $\kappa_2(U) = 3.41 \times 10^4$, which is much smaller than values from the ILU-preconditioned matrices.

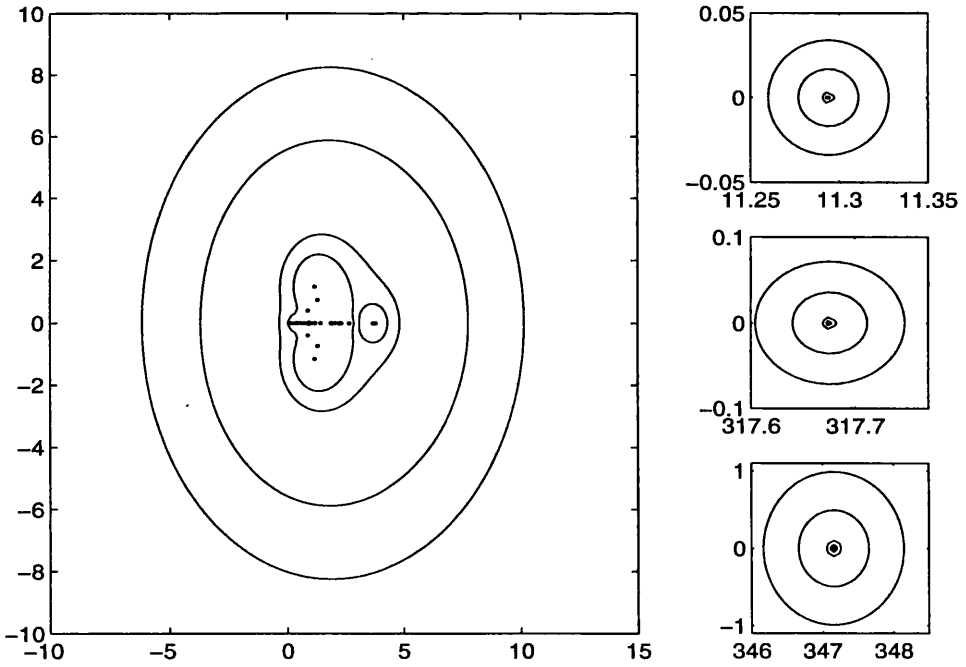

Figure 3-21: *Pseudospectra of matrix H_{36} from GMRES applied to $M_1^{-1}J_5$ from Example 3. The level curves are for $\epsilon = 1 \times 10^{-2}$, 5×10^{-3}, 1×10^{-3} and 5×10^{-4}.*

Although we have compared pseudospectra of matrices H_k for values of k for which GMRES has 'converged', we should also compare pseudospectra of H_k for comparable values of k. Although the convergence criterion (3.10) is satisfied when $k = 36$ for $M_1^{-1}J_5$, we can allow the GMRES algorithm to continue indefinitely. This allows us to generate matrices H_k and their pseudospectra for larger values of k, and compare them with the corresponding examples from ILU preconditioned GMRES. The pseudospectra

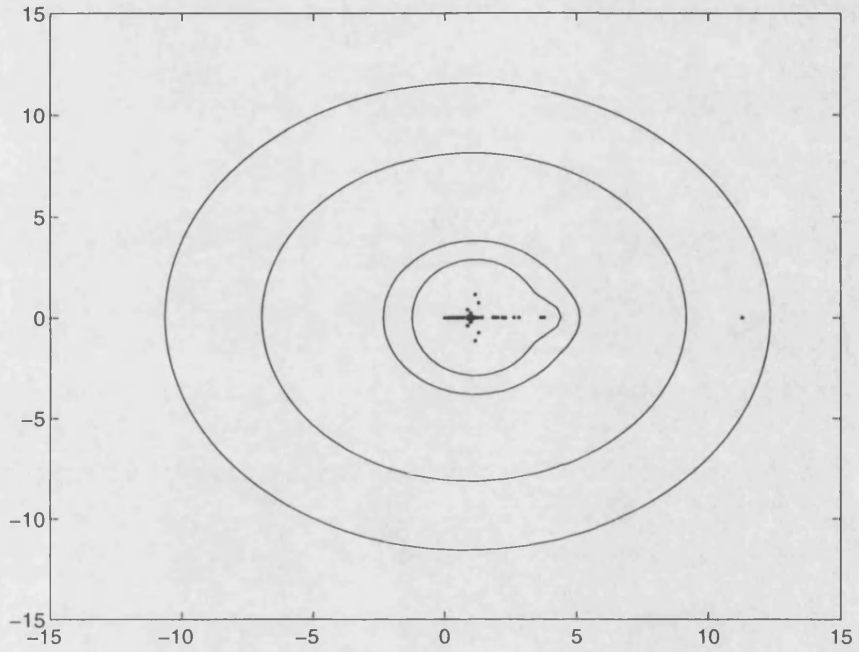

Figure 3-22: *Section of pseudospectra of matrix H_{68} from GMRES applied to $M_1^{-1}J_5$ from Example 3. The level curves are for $\epsilon = 1 \times 10^{-2}, 5 \times 10^{-3}, 1 \times 10^{-3}$ and 5×10^{-4}.*

of H_{68} and H_{150} from GMRES applied to the matrix $M_1^{-1}J_5$ are plotted in Figures 3-22 and 3-23 respectively (the outlying eigenvalues have been omitted for simplicity). If we compare these plots with Figures 3-16 and 3-17, then we can see that whilst the degree of non-normality is increasing with k from 36 (Figure 3-21) through 68 to 150, the matrices are still much less non-normal than the examples from GMRES preconditioned with ILU.

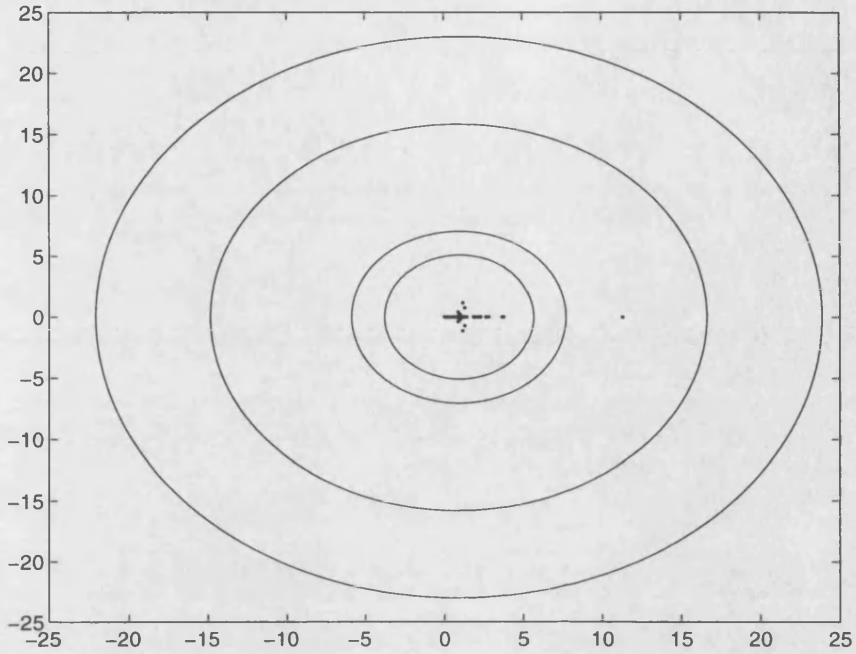

Figure 3-23: *Section of pseudospectra of matrix H_{150} from GMRES applied to $M_1^{-1}J_5$ from Example 3. The level curves are for $\epsilon = 1 \times 10^{-2}, 5 \times 10^{-3}, 1 \times 10^{-3}$ and 5×10^{-4}.*

3.5.6 Numerical Experiments: Example 4

The last example we will examine in this chapter is the application of full LU preconditioning to matrices from Example 4. This is the largest example of the four presented in §3.3, with a dimension of $n = 3787$. The results shown in Table 3.3 show that this preconditioning method is effective for this example too. In spite of larger values for $\|M^{-1}J_l\|$ than were seen when this preconditioner was used for Example 3 matrices (possibly due to the increase in the size of the systems we are solving), we obtain rapid and accurate convergence with the GMRES algorithm. We can again see how the majority of the eigenvalues of the preconditioned system are in a tight cluster around 1, around which we can place a disc centred on $c = 0.99$ with a radius $\rho = 0.025$. Applying Corollary 2.2 gives us the result that

$$\|e_k\|_2 = O\left(\frac{0.025}{0.99}\right)^k.$$

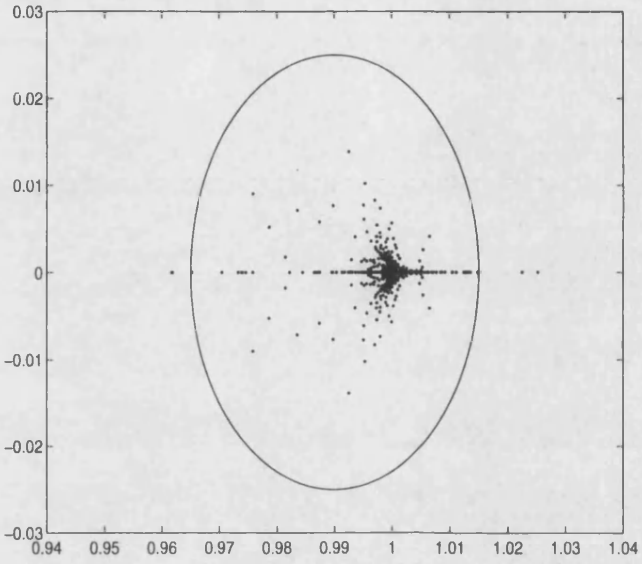

Figure 3-24: *Section of the spectrum of Jacobian matrix J_2 preconditioned with exact inverse of Jacobian matrix J_1 from Example 4. Plot shows a disc of radius $\rho = 0.025$ centred at $c = 0.99$ enclosing the majority of the eigenvalues in this cluster.*

l	k	$\|M_1^{-1}J_l\|$
2	10	1.25×10^4
3	9	8.77×10^3
4	4	3.29×10^3

Table 3.3: *Results of using $M_1 = J_1$ as a preconditioner for GMRES applied to Jacobian matrices from Example 4. Table shows the value of k required for GMRES residual norm estimate to be less than 1×10^{-8} and the value of $\|M_1^{-1}J_l\|$.*

We can use this result to derive an approximate bound for the error. Figure 3-25 shows the norm of the error and a bound obtained by multiplying $(\frac{0.025}{0.99})^k$ by a constant large enough to ensure the bound is larger than the error for all k. We can see from this

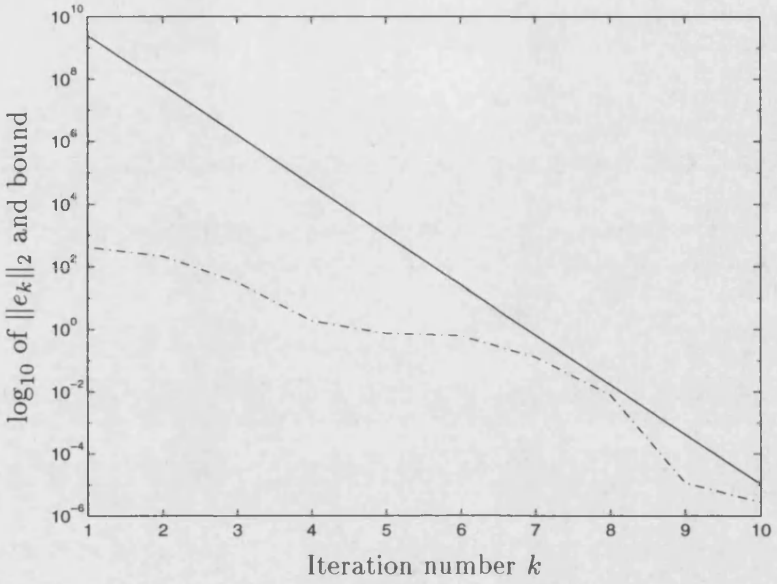

Figure 3-25: $\log_{10}$ *of the error norm and the approximate bound for GMRES applied to* J_2 *from Example 4 preconditioned with the exact inverse of* J_1.

figure that the bound shows reasonable agreement with the observed rate of convergence, especially as the iteration nears termination. At this point, we would expect the contribution to the error from the small eigenvalue terms (see §2.6.3) to be small, and that the convergence would be governed by the remainder of the spectrum, which would explain the good agreement at this stage.

We can calculate the eigenvalues of the upper Hessenberg matrix H_{10} to see how they compare with the spectrum of the preconditioned matrix $M_1^{-1}J_2$, again calculated with MATLAB. The large extremal eigenvalues at $\lambda_{3787} \approx 18$ and $\lambda_{3786} \approx 1.47$ are approximated well, as are the small ones at $\lambda_1 \approx 0.52$, $\lambda_2 \approx 0.61$ and $\lambda_5 \approx 0.89$. However, the GMRES algorithm terminates with no approximation to λ_3 or λ_4, both near 0.71. If we examine the cluster of eigenvalues around 1, we see that they have been approximated by five eigenvalues of H_{10} (see Figure 3-26), again indicating that the cluster required few GMRES iterations to be resolved. Lastly, if we examine the pseudospectra of H_{10} (Figure 3-27), we see that it (and hence the preconditioned matrix) does not suffer

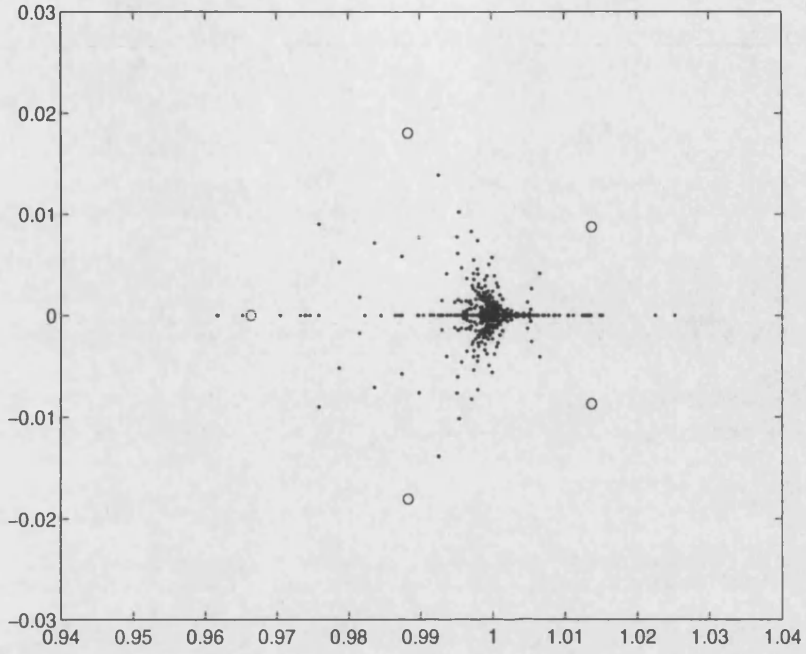

Figure 3-26: *The cluster of eigenvalues of the preconditioned matrix $M_1^{-1}J_2$ from Example 4 (dots) and the eigenvalues of H_{10} from the GMRES algorithm applied to this matrix (circles).*

from a large degree of non-normality. We can therefore conclude that the exact inverse preconditioner works well for the matrices from Example 4.

3.6 Conclusions

In this chapter, we have introduced the concept of preconditioning, and shown why it is necessary for matrices that arise from SPEEDUP. First, we examined the popular ILU preconditioning method and several variants of it, and results have shown that they are largely ineffective for linear systems from SPEEDUP problems. This can be attributed to four reasons: Firstly, although the distribution of the eigenvalues is improved by the preconditioning, it is not sufficiently improved to allow us to apply any results from Chapter 2, or see any significant improvement in the numerical performance of preconditioned GMRES. Secondly, we have shown by means of pseudospectra that the preconditioned matrices suffer from non-normality, which can cause poor convergence of

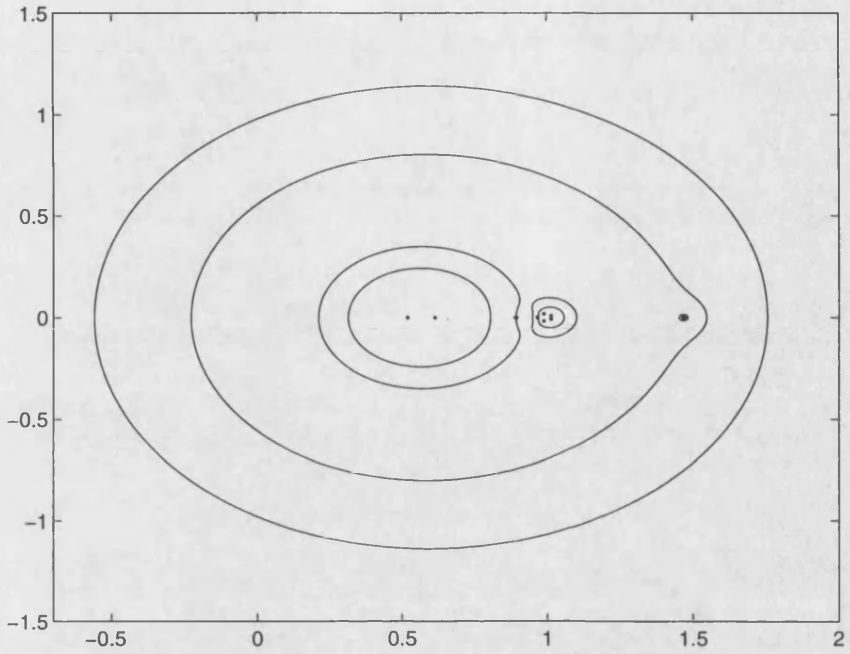

Figure 3-27: *Section of the pseudospectra of the matrix H_{10} arising from the GMRES algorithm applied to the preconditioned matrix $M_1^{-1}J_2$ from Example 4. The level curves are for $\epsilon = 1 \times 10^{-2}$, 5×10^{-3}, 1×10^{-3} and 5×10^{-4}.*

iterative methods regardless of the distribution of the spectrum. Thirdly, the presence of very small eigenvalues results in poor agreement between the GMRES residual and the actual error corresponding to the approximate solution, which could present problems in determining when terminating the algorithm. Finally, we saw how the poor numerical properties of the preconditioned systems yielded poor residual norm approximations, which again presents problems in deciding when to terminate the GMRES process.

We have shown that by preconditioning a sequence of Jacobian matrices (1.1) by an exact inverse of an initial matrix, we can obtain faster and more accurate convergence of the GMRES algorithm than if we had used an ILU variant preconditioner. The problems encountered with the ILU preconditioning are generally avoided with this approach: Although small eigenvalues occur, they are generally not as small as those encountered with ILU, thus whilst still yielding error norms larger than residual norms, the agreement is better than with ILU. The distribution of the spectrum of the preconditioned matrices allows us to apply results from Chapter 2, yielding predicted rates of convergence that agree well with observed results. The pseudospectra analysis shows

that the preconditioned matrices are not as non-normal as those from the ILU preconditioned case. These systems do not share the poor numerical properties of the ILU preconditioned systems, and produce reliable residual norm estimates, allowing accurate termination of the GMRES algorithm. This method has the advantage of reusability, in that we can use one preconditioner for many different linear systems, and once the preconditioner has been calculated, it is cheap to apply, requiring $2(k+1)$ backsolves for k GMRES iterations.

In the next chapter, we will derive a robust algorithm for solving sequences of Jacobian problems (1.1), and test its effectiveness on some SPEEDUP integration runs.

Chapter 4

GMRES as a black-box solver in SPEEDUP

4.1 Introduction

In this chapter we present results and conclusions for the implementation of preconditioned GMRES as an alternative to the current linear solvers in SPEEDUP. GMRES is treated as a 'black-box' solver that provides approximate solutions x_k to a sequence of linear equations (1.1),

$$J_l x = b_l, \;\; l = 1, 2, \ldots$$

such that x_k is accepted only if the preconditioned residual norm

$$\|M^{-1} r_k\|_2 = \|M^{-1}(b_l - J_l x_k)\|_2$$

is less than some fixed tolerance. This sequence arises from the SPEEDUP integration procedure described in Chapter 1.

We discuss the need for a robust solver to guarantee that the integration will run to completion, and derive the Full-LU preconditioned GMRES (FLUGMR) algorithm used in the numerical experiments presented later in the Chapter. We discuss the specifics of the implementation, including the direct solver used to calculate the preconditioners.

Finally, we present numerical results from a number of SPEEDUP flowsheets. These are used to determine the optimal values for various parameters used in the FLUGMR algorithm, and to compare the performance of the GMRES solver with the direct solvers currently in use in SPEEDUP.

4.2 Solver Requirements

The first requirement for such a solver should be robustness rather than performance: A solution method is of limited use if it breaks down or produces an inaccurate result at some stage of an integration, no matter how fast its performance. We must therefore ensure that our preconditioned method produces reliable approximate solutions at every linear solve. We saw in Chapter 3 that the Jacobian matrices from even simple SPEEDUP flowsheets are very ill-conditioned, and efficient preconditioning is required to produce a sufficiently accurate solution within a reasonable number of iterations. Due to the ill-conditioning of the Jacobian matrices, we need to be careful in our choice of preconditioner for two reasons:

- We know from results presented in Chapter 2 that convergence of the GMRES algorithm is related to the distribution of the spectrum, $\sigma(A)$, or in the preconditioned case, $\sigma(M^{-1}A)$. Thus we require the preconditioned matrices to have eigenvalues suitably distributed to ensure rapid convergence. From Chapter 3, we know that by using a suitable preconditioner, the eigenvalues of the preconditioned system can be distributed in tight clusters, or in clusters with small numbers of outliers. From Theorems 2.7 and 2.9, we can therefore expect a sufficiently rapid rate of convergence by using such preconditioners.

- The information usually used to terminate the GMRES algorithm is the norm of the (preconditioned) residual r_k, which is available at no cost during the iteration. We recall from §2.4 that the error can still be large when the residual is small if the error possesses a large component in an eigendirection corresponding to a small eigenvector of the (preconditioned) matrix. Thus care needs to be exercised in deciding when to terminate the iteration, as our preconditioned systems may

possess small eigenvalues.

From Chapter 3, the most successful preconditioner we considered was a sparse LU factorisation of a previous Jacobian matrix from the current integration run, i.e.

$$M^{-1} = J_{old}^{-1}.$$

The experiments conducted in Chapter 3 with Jacobian matrices from SPEEDUP showed that a factorisation of J_1 can be used to successfully precondition several subsequent Jacobians J_l, $l = 2, 3, \ldots$. We would expect the performance of the preconditioner to decrease as the Jacobian J_l changes from the the initial Jacobian J_1.

This suggests that to attempt to avoid the first of the two problems outlined above, we should not rely on a single preconditioner for a whole integration, but use several preconditioners generated at different stages throughout the integration. There are two possible ways to decide when to generate a new preconditioner. The first is to generate a new preconditioner after a fixed number of linear solves have been performed. This choice would require the GMRES solver to be able to solve a system for a given preconditioner, regardless of the suitability of the preconditioner. This may require a long subspace length m or a large number of restarts r in order to achieve convergence, which could be prohibitively expensive.

The second approach is to prescribe a fixed subspace length m and limit the number of restarts r to a maximum of $r_{\max}$, and generate a new preconditioner when these two limits are reached. Using this method, we can ensure that a new preconditioner is generated when the performance of the linear solver deteriorates, whilst avoiding unnecessary calculations when the current preconditioner is performing well. This method also fits in well with the manner in which the behaviour of a SPEEDUP flowsheet changes. The disturbances, as described in Chapter 1, that can require many refactorisations by a direct method, do not necessarily happen at regular intervals. If we assume that such changes in a flowsheet are likely to warrant calculation of additional preconditioners, then this second approach is more appropriate.

This second choice therefore leads to the following algorithm:

Algorithm 4.1 *FLUGMR - Full LU preconditioned GMRES: Used to solve system of linear equations (1.1)*

1. **Start**: Calculate $M^{-1} = J_1^{-1}$. Set $x = M^{-1}b_1$, $l = 2$.

2. **Iterate**: Until integration is completed over the required time period, apply GMRES(m) to

$$M^{-1}J_l x = M^{-1}b_l.$$

If converged then:

(a) Accept x as solution, set $l = l + 1$, else,

(b) If $r \leq r_{\max}$ then restart:

$$x_0 \leftarrow x_m,$$

else calculate new preconditioner:

$$M^{-1} = J_l^{-1}.$$

Goto 2.

Of course, when we refer to M^{-1} and J_l^{-1} in the above algorithm, we do not calculate these inverses exactly, rather use some factorisation method (as in Chapter 3), details of which will be discussed in the following section.

The values of m and $r_{\max}$ govern how the iterative solver will behave: The larger these are, the fewer new preconditioners will be calculated during the iteration. It is therefore likely that there is an optimal value for these two parameters that will yield the fastest solution time for a particular flowsheet.

4.3 Implementation of FLUGMR

4.3.1 LU factorisation

The calculation of exact inverses of general large, sparse matrices is a nontrivial task. Instead, it is usual to construct a factorisation of a matrix such that the action of the inverse on a vector can be calculated. The most common factorisation for non-symmetric matrices is the LU factorisation with partial pivoting,

$$PA = LU, \tag{4.1}$$

where L and U are lower and upper-triangular matrices respectively, and P is some permutation matrix. Whilst for dense matrix problems partial pivoting is a sensible approach to achieving a stable factorisation, when this method is applied to sparse matrices ([22]) large amounts of fill-in can occur in the factorisation, causing poor performance due to the extra arithmetic that the fill-in introduces, and increases the storage requirements of the factorisation.

One way of reducing fill-in is to use *threshold pivoting.* With partial pivoting, pivots are chosen to satisfy

$$|a_{k,k}^{(k)}| \geq |a_{i,k}^{(k)}|, \, i \geq k,$$

where $a_{i,j}^{(k)}$ are the elements of the unfactored submatrix $A^{(k)}$ arising from the kth step of the factorisation. For threshold pivoting, this is replaced with

$$|a_{k,k}^{(k)}| \geq u|a_{i,k}^{(k)}|, \, i \geq k, \, 0 < u \leq 1. \tag{4.2}$$

Note that for $u = 1$, the choice reverts to the normal partial pivot choice. This allows a choice of pivots, and can be applied with a Markowitz criterion [45] to minimise the number of rows and columns changed by the pivot, thus reducing the amount of fill-in. For a thorough treatment of factorisation methods for general sparse matrices, see [22].

SPEEDUP currently uses both the Harwell MA28 [19] and MA48 [23] suites of subroutines as direct linear solvers. Both of these employ techniques similar to the threshold

pivoting approach outlined above. These routines are designed to calculate factorisations of matrices with similar sparsity patterns cheaply, using information from the factorisation of an initial matrix. There are three distinct stages to these routines, which we will denote by ANALYSE, FACTORISE and SOLVE. The operation of the subroutines can then be characterised as follows:

- The ANALYSE phase will examine a matrix and determine from the position and values of the non-zero entries, a set of potential pivots that satisfy the threshold condition. This is most expensive phase of the process, but the calculations performed can be used to generate factorisations of matrices with similar sparsity patterns. Ideally, ANALYSE will only called once for a large number of subsequent factorisations.

- The FACTORISE phase will generate a factorisation of the matrix from the information from ANALYSE and the values of the non-zero entries. Two types of FACTORISE calls are possible. The first type generates a pivot sequence using the ANALYSE data, by performing additional calculations. The second type simply reuses an existing pivot sequence calculated by a previous call to the FACTORISE routine. This type of call can cost as little as 10% of the time of a full FACTORISE call. We will denote this type of call by FASTFAC. Should the old pivot sequence be unsuitable (for example, an element previously used as a pivot could have changed to zero), a new pivot sequence must be calculated using a FACTORISE call. In MA28, this also requires a new ANALYSE phase, but in MA48 this expensive step can be avoided.

- Finally, the SOLVE phase will produce a solution for a given right-hand side vector from the results of the FACTORISE phase with a pair of triangular solves.

The operation of both MA28 and MA48 are summarised in Table 4.1.

The motivation for this project is the slow performance of the existing direct solution methods, caused by the need to perform many FACTORISE (and possibly ANALYSE) operations during short periods of the integration. By using the FLUGMR algorithm with a preconditioner with a three-stage operation as outlined above, we hope to reduce

	Task	
	Factorise initial matrix	Factorise matrix suited to previous pivot scheme
MA28	ANALYSE and FACTORISE	FASTFAC
MA48	ANALYSE and FACTORISE	FASTFAC

	Task	
	Factorise matrix unsuited to previous pivot scheme	Solve linear system
MA28	ANALYSE and FACTORISE	SOLVE
MA48	FACTORISE	SOLVE

Table 4.1: *Summary of the operation of the subroutine suites MA28 and MA48.*

the number of these expensive additional factorisations.

We know from Chapter 3 that we can use a single factorisation as a preconditioner for more than one Jacobian. If we use the MA48 solver to calculate the preconditioner, it may be possible that we will require fewer FACTORISE, FASTFAC and ANALYSE calls than using the direct method alone over the same integration period. We will use many more SOLVE calls, since using the direct solver alone, only one of these is required per linear solve, whereas we will require $k+1$ of these calls, where k is the number of GMRES iterations required to produce an acceptable solution. Recalling from Chapter 3 that one requirement of a preconditioner is that it is cheap to apply, i.e. the operation

$$\bar{x} \leftarrow M^{-1}x \tag{4.3}$$

is not expensive, then we can see that the MA48 routine satisfies this, since (4.3) can be performed using two triangular solves. It is anticipated that the SOLVE process will be the more expensive part of the preconditioned matrix-vector multiplication: If a matrix J_l has NZ non-zeros elements, then to calculate the action of J_l on a vector will require NZ operations. The factorisation will have more than NZ non-zero entries, since some fill-in will occur, thus requiring more than NZ operations. Therefore the action of the preconditioner on a vector will be more expensive to calculate than the action of the matrix on a vector.

4.3.2 Convergence safeguard

To counteract the effect of small eigenvalues producing residual norms much smaller than error norms, we implement a simple safeguard. As the system $J_l x = b_l$ arises in a sequence of Newton steps, then too large an error can result in too large a value for $||F||$. If this occurs, then the nonlinear solver will request another Newton iteration. The GMRES solver will then return a zero value for the Newton step, since the initial (preconditioned) residual will satisfy the convergence criterion, and the Newton iteration will stall. To counter this, if a zero Newton step is encountered, then the convergence criterion will be tightened for one step:

$$\texttt{tol} \leftarrow 10^{-2}\texttt{tol}.$$

This will force the GMRES solver to produce a more accurate solution, and hopefully reduce the error sufficiently to allow the Newton iteration to proceed. The tolerance will be returned to its standard value after that step.

A more sophisticated criterion could be used, based on approximate eigenvalues of the preconditioned system obtained using the Arnoldi process at the foundation of the GMRES algorithm. However, many GMRES iterations may be required to use this approach, as small groups of extremal eigenvalues may not be approximated in by the early GMRES iterations. As we require fast solution times, this approach is probably too expensive to consider.

We now present a series of examples from SPEEDUP flowsheets, using the FLUGMR algorithm as a linear solver.

4.4 Example 1: BTX separation column

The first problem we will examine is one from the SPEEDUP suite of demonstration examples. This system models a Benzene, Tolulene and Xylene extraction column, and has a largest nonlinear block (§1.4.2) of size $n = 927$ (its sparsity pattern is shown in Figure 3-1) from a total DAE system size of 1089. This model is integrated from $t = 0$

to 5 hours in 300 timesteps. With the full Newton solver, this requires around 1220 Newton steps. Real time solution of the BTX model is easily achieved with the current direct methods, so this initial example is used to test robustness and to obtain an initial estimate for optimal values of $r_{\max}$ and m. Since this is a fairly simple example, we will not consider the different types of FACTORISE and FASTFAC calls, but merely count the number of new preconditioners required. The number of new preconditioners and the CPU time taken for the whole integration run for varying values of m and $r_{\max}$ are shown in Table 4.2. These values were obtained by requiring the GMRES solver to obtain approximate solutions such that $\|r_k\|_2 \leq 10^{-8}$.

m	$r_{\max}$	Its	SOL	FFAC	FAC	ANA	CPU secs
3	0	1223	8202	1070	1	1	89.14
3	1	1223	8204	1069	1	1	88.78
3	2	1223	8208	1069	1	1	89.14
3	3	1223	8208	1068	1	1	89.19
4	0	1223	8555	702	1	1	87.33
4	1	1223	8556	701	1	1	87.45
4	2	1223	8560	701	1	1	87.48
4	3	1223	8560	700	1	1	87.59
5	0	1223	8740	357	1	1	85.56
5	1	1223	8744	357	1	1	84.80
5	2	1223	8744	356	1	1	85.45
5	3	1223	8749	356	1	1	85.87
6	0	1223	9209	163	1	1	86.58
6	1	1223	9211	163	1	1	86.92
6	2	1223	9216	163	1	1	86.95
6	3	1223	9221	163	1	1	86.79
7	0	1223	9642	85	1	1	89.18
7	1	1223	9670	84	1	1	89.06
7	2	1223	9675	84	1	1	90.88
7	3	1223	9671	84	1	1	89.31
18	0	1223	16474	5	1	1	140.73

Table 4.2: *CPU timings and number of* SOLVE, FASTFAC, FACTORISE *and* ANALYSE *calls for the BTX example for varying values of* m *and* $r_{\max}$ *using the FLUGMR algorithm. These calculations were performed on an IBM RS6000 3CT machine with 128Mbytes of RAM.*

The optimal value for m for this example is 5, requiring 357 preconditioners for the 1223 nonlinear iterations occurring in the 300 integration steps. The value of $r_{\max}$ seems to have little effect on the solution time. By prescribing a large enough value of m or $r_{\max}$,

it is possible to require only a few preconditioners to complete an integration run for the BTX problem, at the cost of more work performed in the GMRES algorithm. However, this is far slower than the optimal values given in Table 4.2.

The effect of this additional work on the CPU load per timestep can be shown by a simple example. Figure 4-1 shows two plots of CPU/real time for the integration run. The top plot is for $m = 6$ and $r_{\max}$=0 and the bottom is for $m = 18$ and $r_{\max}$=0. Figure 4-1 clearly shows the 'ramping' effect on the solution time of using too few

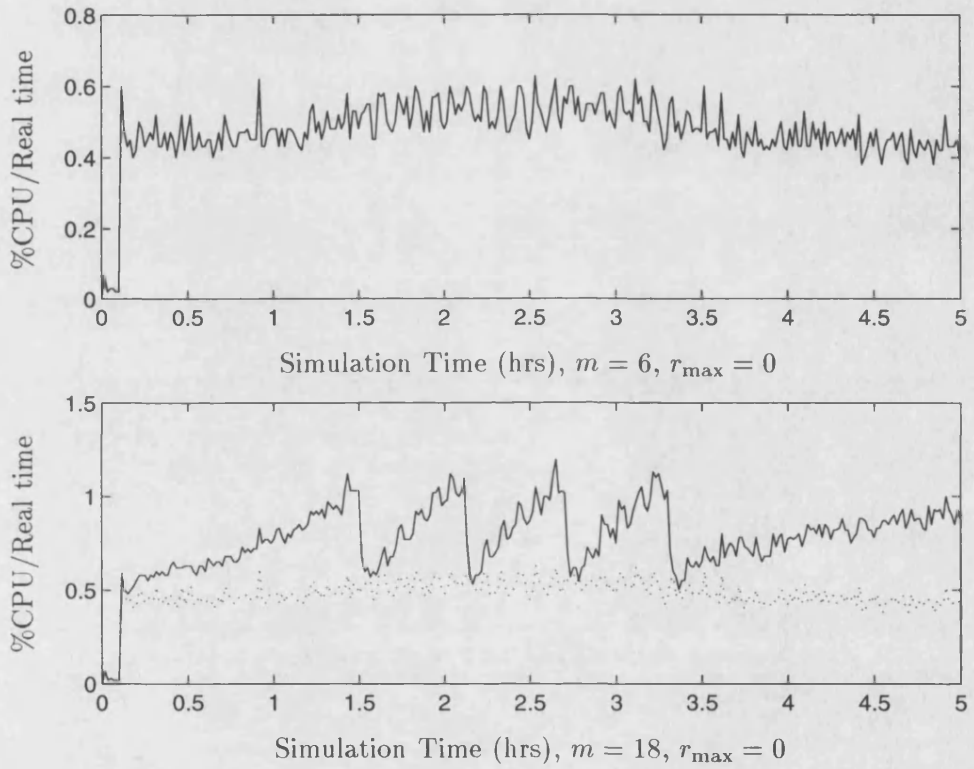

Figure 4-1: *CPU/realtime for the BTX example. Top figure is* $m = 6$, $r_{\max}$*=0, bottom figure is* $m = 18$, $r_{\max}$*=0. The dotted line on the bottom figure is the data for* $m = 6$, $r_{\max}$*=0 for ease of comparison.*

preconditioners, as the GMRES algorithm has to perform more iterations to obtain a sufficiently accurate solution for the $m = 18$ case. The solution time for the $m = 6$ case does not exhibit this effect.

The Brown and Hindmarsh reorthogonalisation criterion (§ 2.7.2) was implemented in the GMRES solver, but with a value of $\alpha = 0.001$ no reorthogonalisations were required.

This indicates that it is not worthwhile trying an expensive Householder GMRES variant on this problem, since loss of orthogonality does not seem to be affecting performance.

This example has shown that the FLUGMR algorithm is able to satisfy our first criterion for an iterative linear solver: It has proved to be robust on a relatively simple SPEEDUP integration. Results have shown that although it is possible to use only a few preconditioner over many timesteps, it is more effective to update the preconditioner more frequently by using small values of m. We will now move on to more realistic SPEEDUP problems for which real-time solution can be problematic.

4.5 Example 2: Plant 3

The second example comes from a case study of a real-world process application performed by Aspentech Consultants. Real-time solution of this problem is not currently possible, with the integration consistently running in excess of 100% of real time. The primary feature of this flowsheet is a reactor producing the chemical 2EH by oxidising propylene, which undergoes a malfunction during the integration. The model has one significant nonlinear block of size 5231, and a total system size in excess of 12,000 DAE's. The integration for this example is over 667 timesteps of 3×10^{-5} hours, totalling 0.02 hours. Table 4.3 shows the number of nonlinear iterations and the number of SOLVE, FASTFAC, FACTORISE and ANALYSE calls required by the FLUGMR algorithm for the large nonlinear block, and the total number of CPU seconds required to complete the integration. The results show that, as was the case for the BTX example, shorter subspace lengths provide the optimal performance, although the $r_{\max}$ parameter seems to play very little part in the performance. Indeed, judging by the number of SOLVE calls made, the average number of GMRES iterations required per iteration seems fairly constant over all the values of $r_{\max}$. In fact for the optimal $m = 6$ case, we see that the averages are 5.801, 5.813 and 5.821 for $r_{\max}$=0, 1 and 2 respectively. Thus we can conclude that, for this example, restarts have a negligible effect on the solution time. Observations show that sometimes, a restart will allow convergence to occur with only one or two extra iterations, thus saving the calculation of a preconditioner. Other times, the $r_{\max}$ limit will be reached, and a new preconditioner will be required anyway.

m	$r_{\max}$	Its	SOL	FFAC	FAC	ANA	CPU secs
5	0	1830	12164	438	22	8	1004.25
5	1	1830	12166	438	23	8	1005.49
5	2	1830	12163	439	23	7	1002.88
6	0	1828	12749	307	21	7	959.06
6	1	1828	12761	307	21	7	960.99
6	2	1825	12754	306	21	7	959.84
7	0	1827	13646	250	19	6	965.87
7	1	1829	13657	249	19	6	965.76
7	2	1829	13662	249	19	7	966.80
8	0	1831	14675	225	19	6	997.46
8	1	1829	14676	223	18	5	988.15
8	2	1827	14677	225	19	6	996.57
9	0	1831	15815	186	20	6	1024.11
9	1	1838	15858	189	20	6	1027.94
9	2	1828	15783	185	20	6	1022.17
18	0	1826	21202	71	10	1	1328.31

Table 4.3: *Results for FLUGMR applied to Example 2 using a full Newton method. The CPU timings are for the entire integration, whilst the solve, factorisation and analyse counts are for the largest block ($n = 5231$). The GMRES tolerance was set to 10^{-10}. These results were calculated on an IBM RS6000 3CT machine with 128Mbytes of RAM.*

It appears that the combination of these effects for the differing values of $r_{\max}$ result in similar numbers of SOLVE calls and preconditioners, and therefore similar solution times.

We can examine exactly how the work is divided between the different routines by performing profiling on the SPEEDUP executable program. This will show what percentage of the solution time was spent in the individual routines. Table 4.4 shows the percentage of CPU time used by each of the top four routines for $r_{\max}$=0, 1 and 2. Note that the totals shown in the table include calls for smaller blocks also. The number of calls and the percentage of time spent in each routine is fairly uniform for each value of $r_{\max}$. A brief description of each routine is given in the table, but in more detail, the relevant routines are:

- `ma50bd`: This routine is responsible for the FACTORISE and FASTFAC calls. No discrimination between the types of call is possible, so the average value for the time per call is over both types of call.

- **ma50cd**: This is the SOLVE routine, and is used to apply the preconditioner after a matrix-vector multiplication.
- **spmvmu**: This routine calculates the product of a matrix (stored in co-ordinate format) with a vector.
- **pgmrgs**: This routine is the body of the GMRES solver, and performs the MGS operation (using **dotvec**) and the QR factorisation of H_k.
- **dotvec**: This calculates the scalar product of two vectors, and is used during the MGS process.
- **l2norm**: This calculates the l_2-norm of a vector.

r_{max}	Routine	%CPU	msec/call	# calls	Total secs
0	ma50cd (SOLVE)	34.1	19.26	14079	271.16
	ma50bd (FACTORISE/FASTFAC)	22.6	544.26	315	171.44
	pgmrgs (MGS/QRFac)	9.4	56.09	2459	137.93
	spmvmu (M-V product)	6.7	10.12	11611	117.44
	dotvec (scalar prod.)	2.8	1.85	26201	48.26
1	ma50cd (SOLVE)	34.0	19.19	14092	270.45
	ma50bd (FACTORISE/FASTFAC)	22.6	545.02	315	171.68
	pgmrgs (MGS/QRFac)	9.5	57.35	2460	141.08
	spmvmu (M-V product)	6.7	10.06	11623	116.92
	dotvec (scalar prod.)	2.8	1.85	26254	48.47
2	ma50cd (SOLVE)	34.1	19.87	14085	279.87
	ma50bd (FACTORISE/FASTFAC)	22.6	543.82	314	170.76
	pgmrgs (MGS/QRFac)	9.4	57.23	2457	140.61
	spmvmu (M-V product)	6.7	10.10	11619	117.40
	dotvec (scalar prod.)	2.8	1.86	26271	49.00

Table 4.4: *Profile results for FLUGMR applied to Example 2 for $m = 6$ and r_{max}=0, 1 and 2. Figures shown are %age of total CPU time used for each routine, the time for each routine, the number of calls to each routine and the total time for each routine.*

The results show that the optimal subspace value with r_{max}=0 results in the majority of the time (40.8%) being spent on calculating the matrix-vector products, with the calculation of preconditioners taking 22.6%. The modified Gramm-Schmidt and QR factorisation operations take less than 10% of the time.

We can see how increasing the subspace length alters the distribution of the CPU time by profiling a run with a value of $m = 18$. Table 4.5 shows results of such run.

Routine	%CPU time	msec/call	# calls	Total secs
`ma50cd` (SOLVE)	31.8	25.09	34650	869.45
`pgmrgs` (MGS/QRFac)	23.6	165.67	3897	645.63
`spmvmu` (M-V product)	11.8	10.47	30744	322.15
`dotvec` (scalar prod.)	9.9	1.86	145424	270.31
`ma50bd` (FACTORISE/FASTFAC)	3.5	753.02	126	94.88

Table 4.5: *Profile results for FLUGMR applied to Example 2 for* $m = 18$, $r_{\max}$*=0. Figures shown are %age of total CPU time used for each routine, the average time for each call and the number of calls to each routine, and the total time spent in each routine.*

As we would expect, the matrix-vector routines `ma50cd` and `spmvmu`, and the main GMRES routine `pgmrgs` have a much larger percentage of the total CPU usage in the $m = 18, r_{\max} = 0$ case. The extra work needed by the GMRES algorithm is also reflected in the much larger average call time for the `pgmrgs` routine. Another consequence is that the `dotvec` routine also uses more of the CPU time than in the $m = 6$ case, caused by the large increase in the number of calls. With fewer preconditioners required, the routine responsible for the FACTORISE and FASTFAC calls is much lower down the list than for the $m = 6$ case, taking only 3.5% of the total time.

So as expected, the optimal values for m and $r_{\max}$ are determined by a balance between the cost of calculating new preconditioners and the increasing overhead of GMRES(m) as m increases, although the predominant factors with the optimal values are the matrix-vector multiplication and the calculation of preconditioners, not the GMRES algorithm itself.

4.5.1 Disturbances

The effect of disturbances introduced to the flowsheet on the CPU loading can be demonstrated by examining a graph of the CPU-time/realtime ratio for the integration period. Figure 4-2 shows two such graphs, one for the optimal choice of $m = 6$, and another for $m = 8$. It can be seen that the integration causes a higher CPU load during the period from $t = 0.01$ to 0.017, due to an increased level of activity in the model. This corresponds to the malfunction, the accidental closure of a gas feed to the

reactor, occurring during this time period. The majority of the new preconditioners are calculated during this period. Indeed, with a large subspace value of $m = 18$, it is possible to integrate up to to $t = 0.01$ requiring only 4 preconditioners. However, once the more active period is entered, large subspace values are of little assistance, and require almost as many preconditioners than the shorter values.

Figure 4-2 shows the effects of calculating additional preconditioners verses more GMRES work: The dotted line on the bottom graph representing $m = 6$ has a consistently lower 'base level' than the solid line of $m = 8$, especially during the initial time period. This can be attributed to the smaller amount of work done by the GMRES algorithm using the shorter subspace length. The calculation of new preconditioners is shown by the spikes, and these are more frequent for the $m = 6$ case than for $m = 8$.

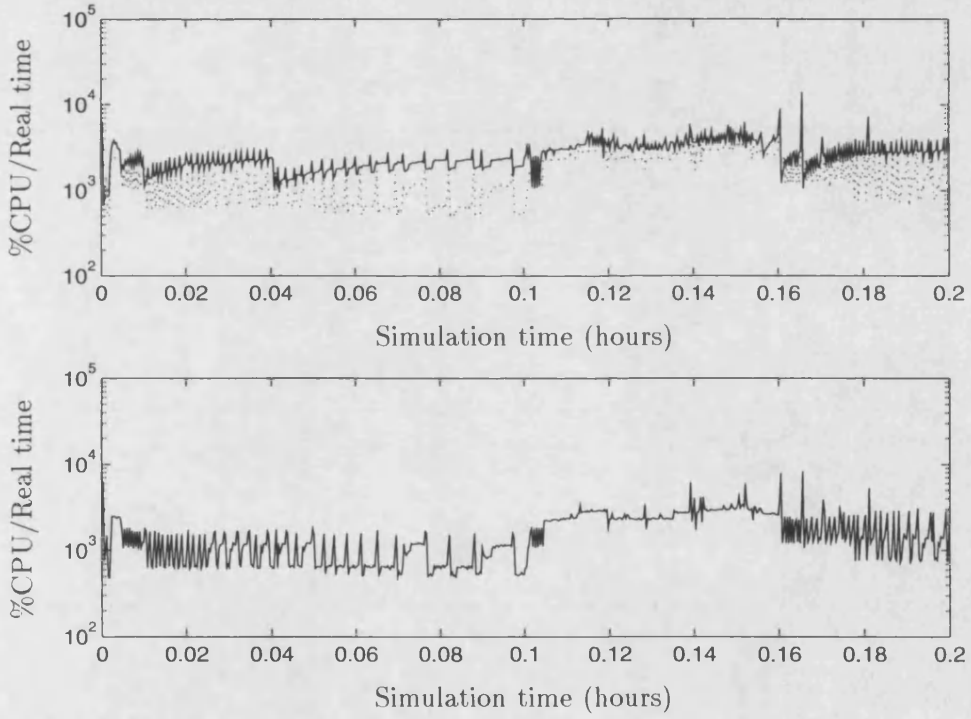

Figure 4-2: *CPU loading for integration of Example 2 model, using FLUGMR as a linear solver. Top figure shows* $m = 6$, $r_{\max} = 0$, *and bottom figure* $m = 8$, $r_{\max} = 0$. *Dotted line on bottom figure shows* $m = 6$, $r_{\max} = 0$ *for ease of comparison.*

4.5.2 FASTNEWTON verses Newton

The results in Table 4.3 were obtained using the standard Newton algorithm. SPEEDUP has another nonlinear solver available, the FASTNEWTON (simplified Newton) option detailed in Chapter 1 (§1.5.2). This option yields better results than a standard Newton method when used in conjunction with the direct solvers in SPEEDUP, by requiring fewer factorisations (as previous Jacobian matrices are used whenever possible) than the standard method, although at a cost of more Newton iterations. However, the motivation for using this approach is only applicable to direct methods. The FLUGMR method should not be affected by changing the Jacobian, since we are not directly dependent on exact factorisations on every Newton step. We would therefore expect a full Newton method to be faster than a simplified Newton method, due to the superior convergence of the full Newton method. We will show that this is indeed the case with the results presented in this section.

Results for the Plant 3 example run using the FASTNEWTON option are shown in Table 4.6, which shows, for varying values of m and $r_{\max}$, the total number of nonlinear iterations required for the large block, the number of FASTFAC, FACTORISE and ANALYSE calls needed, and the total number of CPU seconds required to complete the integration.

Compared to the results shown in table 4.6, it can be seen that, as predicted, using a full Newton method is faster than an Approximate Newton method when using the FLUGMR solver.

4.5.3 Comparison with the Direct Solver

The initial motivation for this project was the poor performance of direct solution methods on large SPEEDUP problems. In spite of preconditioning GMRES with a direct solver, it may still be possible to out-perform the direct solver by requiring fewer of the expensive refactorisations that cause the 'spiking' seen in Figure 1-1. In this section, we will compare the performance of the FLUGMR algorithm with the optimal values of m with the two direct solvers available in SPEEDUP, MA28 [19] and MA48 [23],

m	r_{max}	Its	SOL	FFAC	FAC	ANA	CPU seconds
3	0	3443	13706	468	21	1	1242.39
4	0	3441	14298	350	20	1	1221.27
4	1	3442	14352	350	21	1	1165.53
4	2	3443	14149	351	21	1	1174.65
5	0	3442	14503	334	19	1	1160.83
5	1	3445	14531	333	19	1	1168.78
5	2	3442	14526	332	18	1	1163.21
6	0	3443	16173	306	19	1	1187.12
6	1	3440	16227	309	18	1	1191.33
6	2	3443	16232	306	19	1	1189.03
18	0	3443	33507	118	11	1	1915.46

Table 4.6: *Results for FLUGMR applied to Example 2 using FASTNEWTON method. The GMRES tolerance was set to* 10^{-10}. *These results were calculated on an IBM RS6000 3CT machine with 128Mbytes of RAM.*

both from the Harwell subroutine library.

MA48 is the direct solver used by FLUGMR as a preconditioner. MA28 is unable to perform FACTORISE calls independently of ANALYSE calls - the two must be performed together. MA48 is able to do this, generally resulting in just one ANALYSE call per integration run. However, experiments on some SPEEDUP flowsheets have shown this to have a down-side. With only one ANALYSE call being performed, as the Jacobian matrices evolve over the integration, the FACTORISE and FASTFAC calls require more CPU time to find a suitable factorisation. This resulted in a 'ramping' of the CPU time as the simulation proceeded. The most likely cause of this is that by using 'out-of-date' ANALYSE information, the search for suitable pivots performed by the FACTORISE routine takes more time.

To counter this, MA48 was modified to calculate a rolling average of the time taken to perform the FASTFAC calls. If this average exceeded a user-specified percentage of the first FASTFAC call, the next factorisation would perform an ANALYSE call. This resulted in the elimination of the ramping effect. This modified version is used in FLUGMR. The effects of the modification can be seen in the results of the full Newton method for this example (Table 4.3), which shows more than one ANALYSE call for each set of parameters. For the FASTNEWTON option, although the modification was in place,

no additional ANALYSE calls were needed.

We will therefore compare FLUGMR with MA28, MA48 and the modified MA48, using both Newton and FASTNEWTON nonlinear solvers. Table 4.7 shows the results for these six combinations, along with the fastest result from FLUGMR for comparison. From these results, we can see that FLUGMR produces the best result for solvers using

Method	Its	FFAC	FAC	ANA	CPU seconds
MA28, N	1834	1852	19	19	1174.61
MA28, FN	3442	701	19	19	763.78
MA48, N	1836	1853	18	1	2142.59
MA48, FN	3440	702	21	1	1083.26
MA48 mod, N	1836	1859	24	5	1732.28
MA48 mod, FN	3442	704	22	2	814.90
FLUGMR(6,0), N	1828	307	21	7	959.06

Table 4.7: *Results for various linear solvers applied to Example 2 with Newton (N) and FASTNEWTON (FN) nonlinear solvers. The CPU timings are for the entire integration, whilst the factorisation counts are for the largest block ($n = 5231$). The GMRES tolerance in FLUGMR was set to 10^{-10}. These results were calculated on an IBM RS6000 3CT machine with 128Mbytes of RAM.*

the full Newton method. This can be attributed to the large number of FASTFAC calls required by both direct solvers when using the full Newton option. However, the MA28 and MA48 modified solvers using the FASTNEWTON option both produce faster results. FLUGMR requires the smallest total of FASTFAC and FACTORISE calls by a large margin, but requires far more SOLVE calls. Where the MA28 FASTNEWTON solver only requires 3442 SOLVE calls and the MA48 FASTNEWTON solver 3440, the FLUGMR solver needs 12761, nearly four times as many.

Routine	%CPU time	msec/call	# calls	total secs
`ma30bd` (FAC/FFAC)	30.6	341.23	684	233.40
`ma30ad` (ANA)	21.5	6075.2	27	164.03
`ma30cd` (SOL)	9.5	19.05	3787	72.14
`ma50bd` (FAC/FFAC)	47.8	543.38	716	389.06
`ma50cd` (SOL)	9.2	19.69	3789	74.56

Table 4.8: *Profile results for MA28 and MA48 FASTNEWTON applied to Example 2. Figures shown are %age of total CPU time used for each routine, the average time for each routine, the number of calls to each routine and the total time for each routine.*

If we compare the profiles of the FLUGMR algorithm (Table 4.4) with the profiles of the FASTNEWTON MA28 and MA48 runs (Table 4.8), we can see that whilst for the direct methods, the FASTFAC/FACTORISE routine is responsible for the majority of the solution time, it is the SOLVE calls that take up the majority of the time for the iterative scheme. Unfortunately, the savings made on the factorisations are cancelled out by the extra cost of the SOLVE calls, resulting in slower performance than the optimal combination of the FASTNEWTON approximate Newton method and the MA28 direct linear solver.

4.6 Example 3: Plant 4

The final example in this Chapter is another flowsheet from an Aspentech Consultants problem. The main features of the simulation are a reactor and a pair of distillation columns. The apparatus is again producing 2EH by oxidising propylene. The largest block is of size $n = 25011$, from a total of 59327 variables. The integration consists of 1000 timesteps of 3×10^{-5} hours, totalling 0.5 hours. This problem has a much larger maximum block size than previous problems, so refactorisations are much more expensive to calculate. From Example 2, we know that for FLUGMR to compete with the direct solvers, it must perform substantially fewer of them in order to counteract the cost of the additional SOLVE calls it needs. The results for FLUGMR used with the full Newton method are shown in Table 4.9. As for Example 2, we find that short subspace lengths provide the best performance, and the value of $r_{\max}$ seems to have little effect on the solution time. Again, we can see that large numbers of SOLVE calls are needed. The solution time is greatly in excess of real time, as is shown by the graph of the CPU-time/realtime ration in Figure 4-3. The main point of interest is therefore how the performance compares to the direct solution methods.

4.6.1 Comparison with the Direct Solver

Here we will compare the performance of FLUGMR with the direct solution methods MA28 and MA48. Table 4.10 shows the number of nonlinear iterations, FASTFAC,

m	$r_{\max}$	Its	SOL	FFAC	FAC	ANA	CPU seconds
4	0	3007	16548	400	102	1	6979.06
4	1	3007	16583	450	113	1	6990.17
4	2	3007	16559	454	118	1	6984.48
4	3	3007	16560	451	116	1	6989.84
5	0	3007	17772	187	50	1	6741.49
5	1	3007	17843	169	33	1	6723.64
5	2	3007	17786	212	76	1	6745.62
5	3	3007	17778	175	39	1	6721.82
6	0	3007	18798	97	27	1	6772.22
6	1	3007	18978	99	30	1	6806.36
6	2	3007	18950	103	35	1	6815.84
6	3	3007	18999	103	37	1	6808.86
7	0	3007	20582	58	17	1	7106.14
7	1	3007	20292	61	19	1	7054.31
7	2	3007	20238	60	20	1	7047.61
7	3	3007	20259	59	19	1	7047.47

Table 4.9: *Results for FLUGMR applied to Example 3 using a full Newton method. The GMRES tolerance was set to* 10^{-11}. *These results were calculated on an IBM RS6000 3CT machine with 128Mbytes of RAM.*

FACTORISE and ANALYSE calls, and the total CPU time required by each of the direct solvers in conjunction with both the full Newton method and the FASTNEWTON option. No additional ANALYSE calls were performed when the modified MA48 was used, so these figures are not included here. As the table shows, the FLUGMR algorithm

Method	Its	FFAC	FAC	ANA	CPU seconds
MA28, N	3007	3007	5	5	10704.43
MA28, FN	3386	102	19	19	5957.04
MA48, N	3007	3353	347	1	10962.49
MA48, FN	3385	110	28	1	1917.24
FLUGMR(5,0), N	3007	187	27	1	6722.89

Table 4.10: *Results for various linear solvers applied to Example 3 with Newton (N) and FASTNEWTON (FN) nonlinear solvers. The CPU timings are for the entire integration, whilst the factorisation counts are for the largest block* ($n = 25011$). *The GMRES tolerance in FLUGMR was set to* 10^{-11}. *These results were calculated on an IBM RS6000 3CT machine with 128Mbytes of RAM.*

is again the best linear solver when used in conjunction with the full Newton method. However, the direct solution methods (MA48 in particular) outperform the best iterative

solution when used in conjunction with the approximate Newton method. It can be

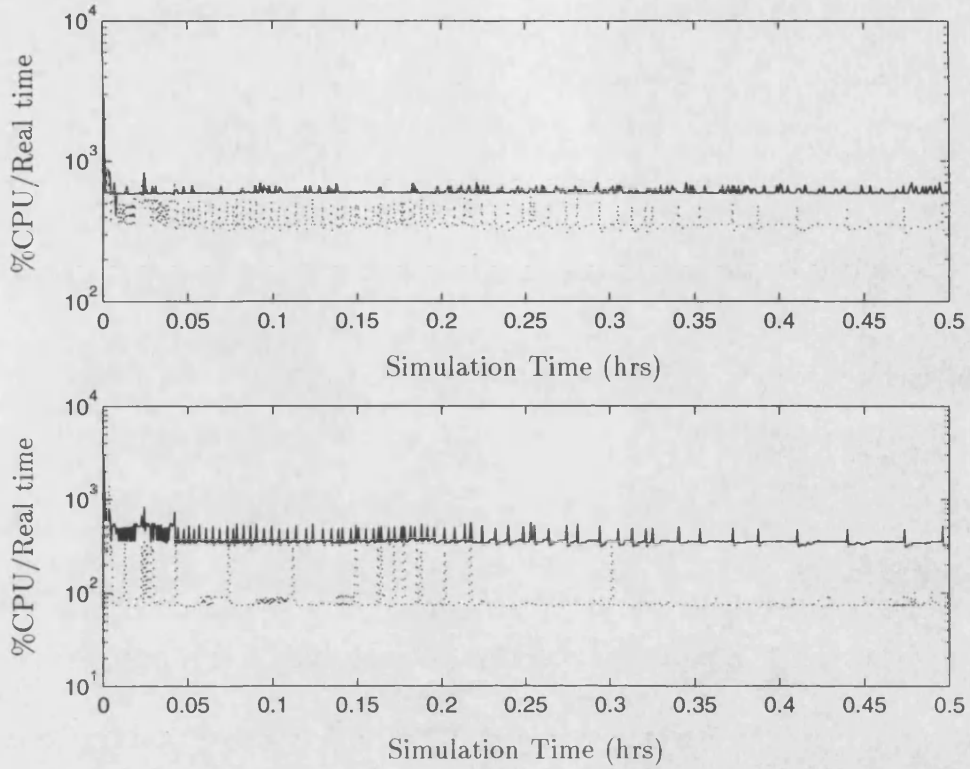

Figure 4-3: *CPU loading for integration of Example 3 model. Figure shows data for FLUGMR-Newton, MA48-Newton and MA48-FASTNEWTON. Top figure shows MA48-Newton (solid line) and FLUGMR(5,1)-Newton (dotted line), bottom figure shows FLUGMR(5,1)-Newton (solid line) and MA48-FASTNEWTON (dotted line).*

seen in Figure 4-3 how the various methods perform compared to real-time. Both the FLUGMR and MA48 methods used with the Newton solver have a fairly consistent CPU loading, with the iterative method being consistently faster than the direct method. The FASTNEWTON option used with MA48 has a much lower base level than the Newton method, with the integration only falling behind real-time when additional factorisations are needed.

4.7 Conclusions

In this chapter, we have taken ideas presented in Chapter 3 and developed them to produce a robust iterative linear solver for use with SPEEDUP. By experimenting with

the solver parameters, we have shown that the fastest solution times are provided by allowing only relatively short subspace lengths ($m = 5$ or 6) before restarting GMRES or calculating new preconditioners. The value of $r_{\max}$, the maximum number of restarts allowed before a new preconditioner is calculated, appears to have little effect on the performance of the algorithm on the larger examples presented in this chapter.

The iterative nature of the solver enables the use of a full Newton method, whereas the direct methods require an approximate Newton method such as SPEEDUP's FASTNEWTON option to produce optimal results. Although FLUGMR uses a direct method for its preconditioner, it was hoped that the iterative method would be faster than the the direct methods alone, by requiring fewer expensive refactorisations of the Jacobian matrices. Examples 2 and 3 have shown that whilst fewer factorisations are required, performance of the iterative solver is not as good as the optimal combination of direct solver and approximate Newton method.

Whilst comparing FLUGMR with MA28 and MA48 using the standard Newton method produces favourable results, the performance is significantly slower when using the FASTNEWTON option. Profiling performed on FLUGMR shows that the calculation of the matrix-vector products required by the algorithm take up the largest proportion of the solution time; more specifically the application of the preconditioner by triangular linear solves. It is the large number of these solves that cause the slower performance when compared to the direct solvers using the FASTNEWTON option. Ultimately, it seems that using an iterative method as a black-box linear solver is not as efficient as the best combination of a direct solver and an approximate Newton method. In the next Chapter, we will examine inexact Newton methods, which are ideally suited to iterative linear solvers, in an attempt to reduce the overall number of linear iterations required to solve a nonlinear system.

Chapter 5

Inexact Newton Methods

5.1 Introduction

In the previous Chapter, we presented the FLUGMR algorithm for the solution of a sequence of linear systems of equations

$$J_l x = b_l, \; l = 1, 2, 3 \ldots \tag{5.1}$$

These systems of equations arose from Newton's method applied to a sequence of non-linear systems

$$F(x) = 0. \tag{5.2}$$

Many variants of the standard Newton method exist. Popular methods include the secant method, Broyden's method and other globally convergent quasi-Newton methods. Details of many of these methods are contained in Dennis and Schnabel [18]. In this chapter, we will consider another variant of Newton's method, the family of so-called *inexact Newton* methods [17], that takes advantage of the fact that we are using an iterative method to solve the linear systems (5.1).

The FLUGMR algorithm produces approximate solutions subject to a fixed convergence

criterion, namely that the kth iterate is accepted as the approximate solution if

$$\|r_k\|_2 \leq \texttt{tol}, \tag{5.3}$$

where r_k is the usual preconditioned residual and `tol` is some fixed value. This takes no account of the Newton process from which the linear system arose. When such linear systems are solved using direct methods, then the linear systems are solved 'exactly', i.e. as close to the exact solution as finite precision arithmetic will allow. When using iterative methods to solve (5.1), we have the freedom to choose how accurate the approximate solution is - we can vary `tol` from one linear solve to the next.

5.1.1 Motivation

The question arises: Why would we wish to do this? The answer lies in the relationship between the linear equations (5.1) and the nonlinear function (5.2). Newton's method is based on the principle that a nonlinear function can be approximated by a linear approximation to it, based on the Taylor expansion of the nonlinear function. This linear model of the function is then solved to generate a new approximate solution, and the process is repeated. It is expected that several such approximations will be required to yield an accurate enough solution to the nonlinear function, with the initial linear solves resulting in quite a large value for $\|F(x)\|$. This is due to the fact that, far away from the solution, the linear model may disagree considerably with the nonlinear function at the exact Newton step. An example of this is shown in Figure 5-1, where $y = x^2$ is compared to the linear model $\bar{y}$ at the point $x = 3$. The figure clearly shows the disagreement between the exact solution ($x = 0$) of $x^2 = 0$ and the solution of the linear model ($x = 1.5$).

The motivation for varying the convergence tolerance is now clear: Why require that the solution of the linear model be highly accurate early on in the Newton process when the nonlinear model will not show the same degree of accuracy? By having a more relaxed convergence tolerance earlier on in the process, time is saved by requiring fewer iterations to be performed by the linear solver, and therefore hopefully decrease the solution time.

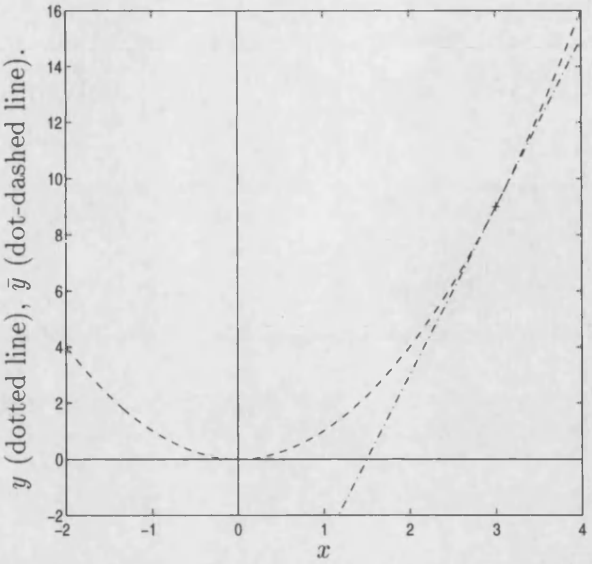

Figure 5-1: *Example of disagreement between nonlinear function and linear model: The function $y = x^2$ (dashed line) has solution $x = 0$, but the linear model $\bar{y}$ based on the point $x = 3$ (dot-dashed line) has solution $x = 1.5$.*

5.1.2 The Inexact Newton algorithm

We can now detail the resulting algorithm, an Inexact Newton method [17]:

Algorithm 5.1 *IN - Inexact Newton Method*

1. **Start**: Choose $x^{(0)}$.

2. **Iterate**: For $k = 1, 2, \ldots$ until 'converged', do:

   Find some $\eta_k \in [0, 1)$ and δx that satisfy

   $$\|F(x^{(k)}) + J(x^{(k)})\delta x\| \leq \eta_k \|F(x^{(k)})\|. \tag{5.4}$$

   Set $x^{(k+1)} = x^{(k)} + \delta x$.

So it is the choice of the η_k terms (not to be confused with the $\eta^{(k)}$ used in Chapter 2) that determine how accurately the algorithm solves the linear model equations. The

role of η_k can be thought of as forcing $\|F(x^{(k)}) + J(x^{(k)})\delta x\|$ to be small in some sense, and thus is often referred to as a *forcing term*, and $\{\eta_k\}$ a *forcing sequence.*

5.1.3 Rates of convergence

The local convergence of such an inexact method is governed by the forcing term. Under assumptions that the function $F(x)$ is continuously differentiable in a neighbourhood of the exact solution $x^{(*)}$ ($F(x^{(*)}) = 0$), the Jacobian $J(x^{(*)})$ is non-singular, and that the Jacobian is Lipschitz continuous at $x^{(*)}$ with constant λ:

$$\|J(x) - J(x^{(*)})\| \leq \lambda \|x - x^{(*)}\|$$

for x near $x^{(*)}$, Dembo, Eisentat and Steihaug [17] show the following results: If $x^{(0)}$ is sufficiently close to $x^{(*)}$, and

$$0 \leq \eta_k \leq \eta_{\max} < 1 \tag{5.5}$$

for each k, then $\{x^{(k)}\}$ converges to $x^{(*)}$ q-linearly in the $\|.\|_*$ norm defined by

$$\|v\|_* \equiv \|J(x^{(*)})v\|$$

for $v \in \mathbb{R}^n$, with asymptotic rate no greater than $\eta_{\max}$. Further, if

$$\lim_{k \to \infty} \eta_k = 0, \tag{5.6}$$

then convergence is q-superlinear, and if

$$\eta_k = O(\|F(x^{(k)})\|), \tag{5.7}$$

then convergence is q-quadratic.

5.1.4 Oversolving

Eisenstat and Walker [26] introduce the concept of *oversolving.* At a point away from the solution, if an η_k is chosen that is too small, then the resulting step may result in

considerable disagreement between F and its local linear model. This oversolving of the Jacobian equation may yield little or no decrease in the value of $\|F\|$, and therefore take us not much nearer the solution. Since we are using iterative methods to perform the solution of (5.1), then requiring an increased level of accuracy will add to the expense of the solution. Thus a less accurate approximation to the full Newton step will be cheaper to calculate and may be just as effective.

5.2 Choosing η_k

We are now faced with the task of choosing the η_k terms to effect the inexact Newton algorithm 5.1. We know from section 5.1.3 it is possible to obtain linear, superlinear or quadratic rates of convergence near to the exact solution $x^{(*)}$, if the sequence $\{\eta_k\}$ is chosen to satisfy the relevant requirements (5.5), (5.6) or (5.7). We will present several choices for the sequence, representing a reasonable cross-section of the literature.

5.2.1 Choice 1

The first choice is very unsophisticated, and was suggested by Cai, Gropp, Keyes and Tidriri [13]:

$$\eta_k = 10^{-4}.$$

As this only satisfies (5.5) and neither of (5.6) or (5.7), then an inexact Newton method with this choice of forcing term will result in linear convergence of the algorithm near the solution. This choice may also require a higher level of accuracy earlier in the Newton iteration than other choices, possibly resulting in oversolving.

5.2.2 Choice 2

Brown and Saad [9] choose

$$\eta_k = \frac{1}{2^{k+1}},$$

resulting in superlinear convergence as both (5.5) and (5.6) are satisfied. For small values of k, this choice allows relatively inaccurate approximations to the exact Newton step to be made, hopefully avoiding oversolving the inexact Newton equation (5.4).

5.2.3 Choice 3

The third choice of forcing term is one suggested by Dembo and Steihaug [16]:

$$\eta_k = \min\left\{\frac{1}{k+2}, \|F(x^{(k)})\|\right\}.$$

Near to $x^{(*)}$ we would expect the $\|F(x^{(k)})\|$ term to be the minimum, and this should result in quadratic local convergence, since it satisfies (5.7). Also, this choice should avoid oversolving early in the Newton process since for small k, η_k will be relatively large.

5.2.4 Choice 4

This choice is one of three presented by Eisenstat and Walker in [26]:

$$\eta_k = \gamma\left(\frac{\|F(x^{(k)})\|}{\|F(x^{(k-1)})\|}\right)^2,$$

with $\gamma \in [0, 1]$. The other alternatives presented in this paper reflect the agreement of F with the local linear model of F at the previous step in an attempt to avoid oversolving. However, these choices are more expensive to evaluate than the one presented above, and Eisenstat and Walker remark that experiments completed in [26] demonstrate that the choice we will use also results in little oversolving in practice. Like choice 3, we can expect local quadratic convergence with choice 4, and Eisenstat and Walker suggest that this choice has the advantage over other choices in that it is not sensitive to the scaling of F.

5.2.5 Choice 5

The final choice is a heuristical one similar to Choice 1, but takes account of the ill-conditioning which results in poor agreement between the norm of the preconditioned residual and the actual achieved error norm (see, for example, Table 3.1). η_k is defined as

$$\eta_k = \frac{\omega}{10^{-k}}$$

where $\omega \in [0, 1]$ is some weight value. Clearly, with $\omega = 0$, this choice reverts back to the standard Newton's method. This choice may suffer from the same drawbacks as Choice 1, but both (5.5) and (5.6) are satisfied, so local convergence should be superlinear. Compared to Choice 2, even for $\omega = 1$ this choice produces smaller forcing terms, resulting in more accurate linear solves throughout the nonlinear iteration.

5.3 Implementation

The FLUGMR algorithm will be used as the linear solver, with the convergence tolerance being set to $\eta_k ||F||$ as per (5.4). Two other measures are included to ensure robust convergence of the algorithm:

5.3.1 Globalisation

The majority of the literature concerning inexact Newton methods advocate not using the IN algorithm 5.1 on its own, but with some form of global convergence scheme; examples are given in [9, 25, 26, 70]. Backtracking is a popular strategy to ensure globalisation, although more elaborate methods based on trust region methods are presented in [25].

The SPEEDUP nonlinear solver has a global convergence strategy implemented in both the full Newton and approximate Newton algorithms. If no reduction in $||F||$ has been observed for a number of iterations (the default is 10), then the solver will return to the best solution point $x^{(\text{best})}$ and take half the Newton (or approximate Newton) step

$\delta x^{(\text{best})}$, and the iteration proceeds with the new step defined as:

$$x^{(k+1)} = x^{(\text{best})} + 0.5\delta x^{(\text{best})}.$$

Such an approach is often called 'damped Newton'. It is this approach that we will use to try to ensure global convergence of the Newton process.

5.3.2 Convergence safeguard

We will also use the same sort of safeguard for the convergence criterion as we did for the black-box solver (§4.3.2) to counteract the possible effects of ill-conditioning in the preconditioned linear system (§4.2): If a zero inexact Newton step is detected and the inexact Newton iteration has not converged, then

$$\eta_k \leftarrow 10^{-2}\eta_k$$

and the inexact Newton procedure is continued.

We will now present results of the five different choices presented in §5.2 applied to various SPEEDUP integration runs from Chapter 4.

5.4 Examples

5.4.1 Example 1: BTX separation column

This is the same flowsheet as Example 1 in Chapter 4 (§4.4). As before, this example will be used to see if the inexact Newton methods are a viable solution method for this fairly simple SPEEDUP flowsheet. The number of nonlinear iterations and the number of SOLVE, FACTORISE, FASTFAC and ANALYSE calls required for the large block ($n = 927$) during the iteration, and the total CPU time to complete the iteration are shown in Table 5.1. Choice 4 is used with values of γ of 0.5, 0.7 and 0.9 (in [26], a value of $\gamma = 0.9$ was found to give best results). Choice 5 is tried with values of ω of

10^{-5}, 10^{-7} and 10^{-9}.

Choice	Its	SOL	FFAC	FAC	ANA	CPU secs
1	1277	5005	0	1	1	69.61
2	3313	15667	3	1	1	172.93
3	2748	12203	10	1	1	142.08
4, $\gamma = 0.5$	2007	10561	39	1	1	119.88
4, $\gamma = 0.7$	2055	10825	39	1	1	122.35
4, $\gamma = 0.9$	2073	10814	39	1	1	122.54
5, $\omega = 10^{-5}$	1374	7480	21	1	1	88.98
5, $\omega = 10^{-6}$	1246	7074	32	1	1	84.22
5, $\omega = 10^{-7}$	1235	7352	42	1	1	86.16
FLUGMR(5,0)						
	1223	8740	357	1	1	85.56

Table 5.1: *Results for various choices of forcing term for an inexact Newton method applied to Example 1. The CPU timings are for the entire integration, whilst the solve, factorisation and analyse counts are for the largest block ($n = 927$). All FLUGMR runs had a maximum subspace length of $m = 5$ and $r_{\max}$=0. These results were calculated on an IBM RS6000 3CT machine with 128Mbytes of RAM.*

As we can see, Choice 1 is easily the most efficient candidate, requiring the fewest SOLVE FASTFAC and FACTORISE calls for the integration. Indeed, only one preconditioner is needed. This results in better performance than the black-box solver.

Choice 5, with an optimal value of $\omega = 10^{-6}$, is roughly similar in performance to the black-box method, requiring fewer SOLVE and FASTFAC calls, but at a cost of more Newton iterations.

Choices 2 and 3 resulted in a large increase in the number of Newton steps, and this is also reflected in the increased number of SOLVE calls. This caused significantly slower performance than the black-box solver.

Choice 4 also resulted in more Newton iterations and SOLVE calls, and performance is also slower than the black-box method, although not to the same degree as for 2 and 3. It was noted that Choices 2, 3 and 4 all required significant numbers of adjustments to the convergence tolerance due to the convergence safeguard (§5.3.2).

5.4.2 Example 2: Plant 3

This example was first considered in Chapter 4 (§4.5). It has a largest nonlinear block of size 5231, and the integration consists of 667 timesteps. The number of nonlinear iterations and the number of SOLVE, FACTORISE, FASTFAC and ANALYSE calls required for the large block during the iteration, and the total CPU time to complete the iteration are shown in Table 5.2. Choice 4 is used with values of γ of 0.5, 0.7 and 0.9 (in [26], a value of $\gamma = 0.9$ was found to give best results). Choice 5 is tried with values of ω of 10^{-3}, 10^{-4}, 10^{-5} and 10^{-6}.

Choice	Its	SOL	FFAC	FAC	ANA	CPU secs
1	2570	16220	56	9	1	1160.74
2	5494	25164	71	9	2	1674.02
3	4527	19888	53	9	1	1415.10
4, $\gamma = 0.5$	3367	16201	81	11	1	1212.92
4, $\gamma = 0.7$	3297	16081	75	11	1	1197.33
4, $\gamma = 0.9$	3494	17240	91	12	2	1247.48
5, $\omega = 10^{-3}$	2151	12941	73	10	1	980.89
5, $\omega = 10^{-4}$	2044	12787	87	11	2	971.87
5, $\omega = 10^{-5}$	1883	12458	116	12	2	942.65
5, $\omega = 10^{-6}$	1838	12643	145	13	2	992.22
FLUGMR(6,0)						
	1828	12749	307	21	7	959.06
MA28 FASTNEWTON						
	3442	3442	1852	19	19	763.78

Table 5.2: *Results for various choices of forcing term for an inexact Newton method applied to Example 2. The CPU timings are for the entire integration, whilst the solve, factorisation and analyse counts are for the largest block ($n = 5231$). All FLUGMR runs had a maximum subspace length of $m = 6$ and r_{max}=0. These results were calculated on an IBM RS6000 3CT machine with 128Mbytes of RAM.*

Unlike the BTX example, Choice 1 is not faster than the standard black-box method for this Example. Although reducing the approximate[1] average CPU seconds per Newton step from 0.52s to 0.45s, and the average number of SOLVE calls per Newton step from 6.97 to 6.31, the increase in the number of Newton steps required to complete the integration results in slower performance.

[1]The average is calculated using the total number of CPU seconds for the integration, which includes the solution of the smaller blocks in the decomposition

Again, Choices 2 and 3 are much slower than the black-box method, requiring many more Newton steps. Choice 4 results in a similar performance, although again, not as bad as either of Choices 2 or 3. The value of γ has little effect on the performance of Choice 4.

So it is Choice 5 that produces the best results for this example, managing a faster solution time than the black-box method with a value of $\omega = 10^{-5}$. However, the performance increase is not large, and the direct method is still significantly faster.

Once again, Choices 2, 3 and 4 all required many adjustments to the convergence tolerance due to the convergence safeguard.

5.5 Conclusions

The results presented in this chapter indicate that whilst faster solution times requiring both fewer FACTORISE/FASTFAC calls and fewer SOLVE calls are possible using inexact Newton methods, the increase in performance is not significant. Therefore, the FLUGMR algorithm is still not as fast as the best direct method. For the real-world problem in Example 2, the performance difference between the black-box method and the best choice was negligible.

The different forcing terms produced varying results, with those resulting in fewest Newton steps producing the fastest results. This can be easily explained: Fewer steps means fewer GMRES iterations and therefore fewer SOLVE calls - the number of FACTORISE/FASTFAC calls appears to be secondary to this in determining optimum performance.

Choice 1 was the best for the BTX example, but for Example 2 was not as good as the ordinary black-box FLUGMR method, in spite of a reduced average time per Newton iteration, as more of these iterations were required. Choices 2 and 3 gave a significant increase in the number of steps for both examples, thus resulting in their poor performances. Choice 4 produced similar results for various values of γ, and was generally better than Choices 2 and 3. However, it is Choice 5 that gave the best results

for Example 2, requiring slightly more Newton iterations than the black-box method, but at a lower cost per iteration. The number of SOLVE calls was similar in both these cases, but the inexact method required fewer FACTORISE/FASTFAC calls.

It seems that it is the attempts to avoid oversolving by the more sophisticated choices that result in the slower performance. This is likely to be because of the observed poor agreement between the size of the preconditioned GMRES residual and the actual error, and therefore the value of $||F(x)||$. Whilst the analysis, which does not take this disagreement into account, predicts a reduction in $||F(x)||$ related to the residual norm, this reduction is not necessarily observed with problems from SPEEDUP flowsheets. The choices that require a tighter tolerance earlier in the Newton iteration (1,5) cause this disagreement to be smaller by insisting on a more accurate approximate solution for a given step. This can be verified by considering the smaller number of adjustments to the tolerance by the convergence safeguard (§5.3.2) for these two choices compared to the other three. Thus whilst the Newton steps are not as cheap as they could be, they are on average cheaper than the steps required for the black-box solver examined in the previous chapter.

The next chapter will consider other iterative methods as an alternative to GMRES, and their performances in SPEEDUP flowsheets examined.

Chapter 6

Other Iterative Solvers

6.1 Introduction

In the previous chapters, we have considered the performance of the GMRES iterative solver. As was mentioned in Chapter 2, there are a number of other Krylov subspace iterative methods for nonsymmetric linear systems. The aim of this chapter is to present some examples of these solvers, and examine their performance on preconditioned SPEEDUP problems. As was mentioned in §2.2, there are three categories of Krylov subspace solvers:

a) Methods based on the normal equations

b) Methods employing orthogonalisation techniques

c) Methods employing biorthogonalisation techniques

GMRES is a type b) method: It produces an l_2-orthogonal basis for $\mathcal{K}_k$, which it uses to produce optimal approximate solutions on the subspace. The disadvantage of this category of method is that the cost of the orthogonalisation process grows with the iteration number. Generally, methods of types a) and c) use short recurrence relationships to generate iterates, and so have a fixed amount of work associated with each iteration.

This means that such methods can be run without being restarted, unlike the GMRES algorithm. We shall first examine type a) methods, based on the normal equations.

6.2 Methods based on the Normal Equations

6.2.1 Forming the normal equations

The conjugate gradient (CG) method [35] is the method of choice for the solution of linear systems (2.1) where the coefficient matrix A is symmetric positive definite (SPD). It has the properties that iterates are generated using a 3-term recurrence relation, and these iterates satisfy an optimality property. Since SPEEDUP matrices are not SPD, we cannot apply CG to these problems directly. However, we can form an SPD system by multiplying (2.1) by A^T:

$$A^T A x = A^T b. \tag{6.1}$$

This has the same solution as (2.1) and can be solved using the CG method. This method is often denoted by CGNR, for residuals, since the A-norm (see §6.2.2) of the CGN residuals is minimised over the Krylov subspace.

The system (6.1) is not formed explicitly, rather the action of A^T on a vector is required. For some applications calculating this action is not possible, but for SPEEDUP problems it is freely available.

This method uses a different Krylov subspace to that used by the other two classes (2.2): Iterates x_k are produced such that

$$x_k \in x_0 + \mathcal{K}_k(A^T A, r_0). \tag{6.2}$$

This means that two matrix-vector products (one with A, the other with A^T) are required at each iteration, compared to a single one for other Krylov methods. The use of the normal equations also means that whereas the convergence of the standard CG method is governed by the spectrum of A (see Kelley, [38]), the convergence of CGN is governed by the spectrum of $A^T A$, namely the singular values of A.

6.2.2 CG iteration

If A is SPD, the conjugate gradient method produces iterates x_k which minimise the A-norm,

$$\|x\|_A = \sqrt{x^T A x}, \tag{6.3}$$

of the error over the space $\mathcal{S} = x_0 + \mathcal{K}_k$ ((6.3) defines a norm since A is SPD). Properties such as finite termination can be proved in a similar manner to those for the GMRES algorithm (see, for example, Johnson [37, §7.3]).

The standard result ([15, 37, 38]) regarding CG convergence relates the kth error norm to the initial error norm:

$$\|x_k - x\|_A \leq 2\|x_0 - x\|_A \left[\frac{\sqrt{\kappa_2(A)} - 1}{\sqrt{\kappa_2(A)} + 1}\right]^k, \tag{6.4}$$

where x is the exact solution to (2.1) and $\kappa_2(A)$ is the standard condition number of A. The eigenvalues of $A^T A$ are the squares of the singular values of A, with eigenvectors given by the right singular vectors of A. We can thus state that the convergence of CGN is governed by the singular values of A. It is easily shown that the condition number $\kappa_2(A^T A) = \kappa_2(A)^2$:

$$\kappa_2(A^T A) = \frac{\lambda_n^{(A^T A)}}{\lambda_1^{(A^T A)}} = \left(\frac{\sigma_n^{(A)}}{\sigma_1^{(A)}}\right)^2 = \kappa_2(A)^2,$$

and so we can obtain an expression for the rate of convergence of CGN,

$$\|x_k - x\|_{A^T A} \leq 2\|x_0 - x\|_{A^T A} \left[\frac{\kappa_2(A) - 1}{\kappa_2(A) + 1}\right]^k, \tag{6.5}$$

compared to (6.4) for standard CG. The main point of note is that we have lost the square root in the convergence coefficient. Thus for even moderate values of $\kappa_2(A)$, we can expect to see very slow convergence for CGN.

6.2.3 Reusing preconditioners

We saw in Chapters 3 and 4 that it is possible, desirable in fact, to use a preconditioner to solve many subsequent linear systems. We can show that it is likely that preconditioners applied to the normal equations will not possess this property when applied to SPEEDUP problems. Recall from §3.5.1 that we considered the sequence of Jacobian matrices in the form

$$J_l = J_1 + D_l, \, l = 2, \ldots,$$

then if the value of $\|M^{-1}D_l\|$ is 'small' for a preconditioner M derived from J_1 (for example, an exact factorisation of J_1), then this preconditioner can be expected to yield good convergence for the iterative method to which it is applied. We can reasonably expect $\|M^{-1}D_l\|$ to be small if $\|D_l\|$ is small (although results from Chapter 4 have shown that rapid convergence is possible even with relatively large values of D_l).

If we consider the matrices $J_l^T J_l$ in the same manner as above, we have

$$J_l^T J_l = J_1^T J_1 + D_l^T J_1 + J_1^T D_l + D_l^T D_l. \tag{6.6}$$

If we apply a preconditioner generated from J_1 to the normal equations (6.6) (recalling that left and right preconditioning is required to preserve symmetry), then we obtain an equivalent quantity to $\|M^{-1}D_l\|$ in the form

$$\|\mathcal{M}_1(D_l^T J_1 + J_1^T D_l + D_l^T D_l)\mathcal{M}_2\|,$$

where $\mathcal{M}_1$ and $\mathcal{M}_2$ represent the preconditioning. As before, we can reasonably expect this quantity to be small if $\|D_l^T J_1 + J_1^T D_l + D_l^T D_l\|$ is small. However, this quantity will not necessarily be small even if D_l is small. If we look at Example 3 from §3.3, we can see that this value is several orders of magnitude larger than $\|D_l\|$, suggesting that preconditioners generated from J_1 may give poor performance with this algorithm. This confirmed in §6.2.4, which shows the poor performance of CGN applied to SPEEDUP Jacobians when preconditioned with a factorisation of a previous Jacobian.

l	$\|D_l\|$	$\|D_l^T J_1 + J_1^T D_l + D_l^T D_l\|$
2	1.64×10^2	8.49×10^4
3	3.37×10^2	9.04×10^6
4	1.64×10^2	8.49×10^4
5	1.32×10^3	5.11×10^7
6	1.64×10^2	8.49×10^4

Table 6.1: Values of $\|D_l\|$ and $\|D_l^T J_1 + J_1^T D_l + D_l^T D_l\|$ from Jacobian matrices from Chapter 3 Example 3.

6.2.4 Numerical Experiments

We can now examine the effectiveness of using preconditioners generated from previous Jacobians in conjunction with CGN iteration. To preserve symmetry, we must apply the preconditioning matrix M twice, once to each matrix-vector product. This gives us the preconditioned system

$$(M^{-1}A)^T M^{-1}A = A^T M^{-T} M^{-1} A = A^T M^{-T} b. \tag{6.7}$$

Recalling (6.5), then we can examine the condition number of our preconditioned systems to see what sort of effect the preconditioning may have on the convergence of the algorithm. Using the preconditioner that was found to be most effective in Chapter 3, i.e. a factorisation of a previous Jacobian, it can be seen that whilst conditioning is improved slightly, we can still expect poor convergence from the CGN algorithm when applied to these examples. Table 6.2 shows the condition numbers for Example 3 from Chapter 3. Some convergence curves are shown in Figure 6-1 for three Jacobians from Example 3, Chapter 3.

l	$\kappa_2(J_l)$	$\kappa_2(M_1^{-1}J_l)$
2	2.58×10^{13}	8.52×10^{10}
3	3.30×10^{14}	1.39×10^{14}
4	2.75×10^{14}	7.89×10^{11}
5	1.41×10^{12}	7.17×10^{12}
6	7.58×10^{13}	2.04×10^{11}

Table 6.2: *Values of $\kappa_2(M_1^{-1}J_l)$ from Jacobian matrices from Chapter 3 Example 3.*

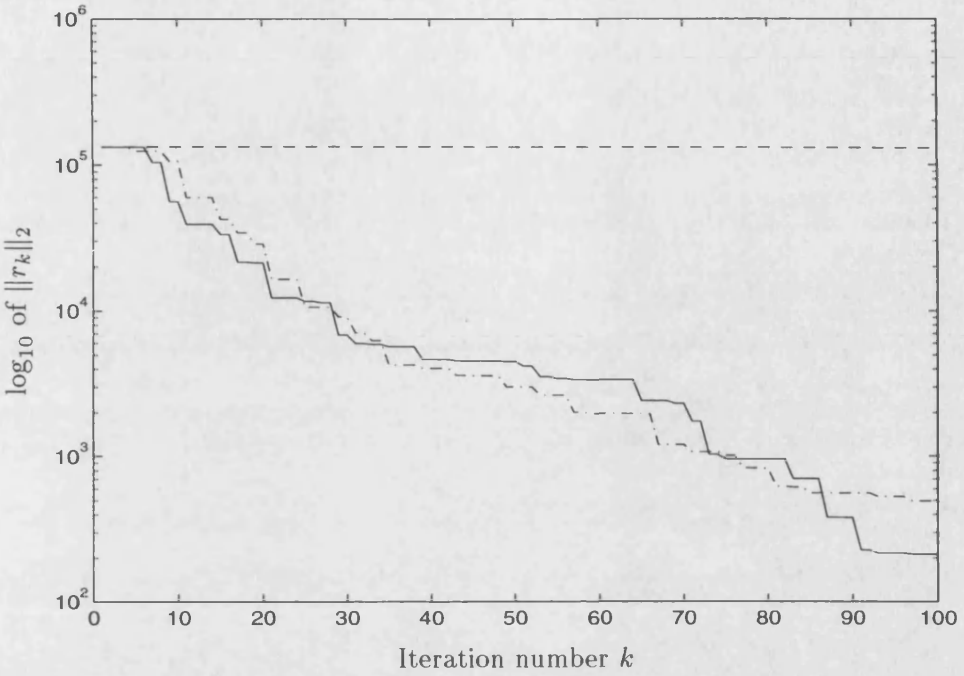

Figure 6-1: *Convergence curves for CGN applied to Jacobians from example 3, Chapter 3, preconditioned with the exact inverse of* J_1. *Solid line:* J_2, *dash-dotted line:* J_4, *dashed line:* J_5.

This figure shows the poor performance that was predicted by the condition numbers. Whilst GMRES was converging in 3, 7 and 36 iterations respectively for Jacobians J_2, J_4 and J_5, we see that CGN is has not produced a sufficiently accurate approximate solution after 100 iterations for each Jacobian, and for J_5 has not made a significant reduction in the residual norm at all. From these results, we can conclude that CGN is not an effective solver for problems arising from SPEEDUP integration runs. This is not surprising, as solving the normal equations is generally considered to be a poor method for non-symmetric linear systems (for special cases where it is an effective approach, see for example Freund *et al.* , [28]).

6.3 Biorthogonalisation methods

6.3.1 The Lanczos method

So far, we have seen Krylov methods which use orthogonalisation to produce optimal iterates (GMRES, Chapter 2), but at a cost of requiring increasing amounts of arithmetic and workspace as the iterative process proceeds, and CG-type methods applied to the normal equations (6.1) whose iterates are uniformly cheap to produce but whose convergence is too slow to be effective.

Methods that employ *biorthogonalisation* techniques are a compromise: Their iterates do not satisfy a strict optimality property like their GMRES counterparts, instead they satisfy some quasi-optimality property on the same Krylov subspace. However, like CG methods, the iterates are available at a uniform cost per iterate.

These methods are based on ideas first developed in the non-symmetric Lanczos method, proposed by Lanczos in 1950 [40]. This algorithm reduces a general matrix $A \in \mathbb{R}^{n\times n}$ to tridiagonal form. The algorithm starts with two non-zero n-dimensional vectors v_1 and w_1 and generates sets of basis vectors $\{v_i\}$ and $\{w_i\}$ for the two Krylov subspaces $\mathcal{K}_k(v_1, A)$ and $\mathcal{K}_k(w_1, A^T)$ respectively, subject to the *bi-orthogonality condition*

$$w_i^T v_j = \delta_{i,j}. \tag{6.8}$$

These bases have the property that they can be constructed with only three-term recurrences. If we denote the coefficients of the recurrence relationships by α_i, β_i and γ_i, then we can define the tridiagonal matrix $H_k^{(e)} \in \mathbb{R}^{(k+1)\times k}$ as

$$H_k^{(e)} = \begin{bmatrix} \alpha_1 & \beta_1 & 0 & \dots & 0 \\ \gamma_2 & \alpha_2 & \ddots & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & \beta_k \\ \vdots & & \ddots & \gamma_k & \alpha_k \\ 0 & \dots & \dots & 0 & \gamma_{k+1} \end{bmatrix}, \tag{6.9}$$

and $H_k \in \mathbb{R}^{k\times k}$ as

$$H_k = [I_k \;\; 0]H_k^{(e)}. \tag{6.10}$$

If we write the two bases in matrix form,

$$V_k = [v_1 \; v_2 \; \ldots \; v_k], \quad W_k = [w_1 \; w_2 \; \ldots \; w_k],$$

then we can relate A, V_k, W_k and H_k by

$$\begin{aligned} AV_k &= V_k H_k, \\ A^T W_k &= W_k H_k^T, \end{aligned} \tag{6.11}$$

(cf. (2.9)) and the biorthogonality condition (6.8) can be written

$$W_k^T V_k = I_k,$$

where I_k is the k dimensional identity matrix.

6.3.2 Biconjugate Gradients

The Biconjugate Gradient algorithm (BCG) is a Krylov subspace method originally proposed by Lanczos in 1952 [41]. Like the Lanczos method described above, it produces bases for two Krylov subspaces subject to a biorthogonality condition. The algorithm is started with two vectors $r_0 = b - Ax_0$ and $\hat{r_0}$, where $\hat{r_0}$ is a user-supplied vector, often set to r_0. Sequences of residuals $\{r_k\}$ and $\{\hat{r}_k\}$ such that $r_k \in \mathcal{K}_k(A, r_0)$ and $\hat{r_k} \in \mathcal{K}_k(A^T, \hat{r}_0)$, and subject to the biorthogonality (Galerkin) condition

$$r_k^T w = 0 \text{ for all } w \in \mathcal{K}_k(A^T, \hat{r}_0). \tag{6.12}$$

There is a residual polynomial $\bar{p}_k \in \mathcal{P}_k$ such that

$$r_k = \bar{p}_k(A)r_0 \text{ and } \hat{r}_k = \bar{p}_k(A^T)\hat{r}_0. \tag{6.13}$$

Since the BCG iterates are in the same subspace as the GMRES iterates, Lemma 2.1 gives us that

$$\|r_k^{\mathrm{GMRES}}\|_2 \leq \|r_k^{\mathrm{BCG}}\|_2, \tag{6.14}$$

so GMRES will always reduce the residual norm more rapidly than BCG provided it is not restarted: However, if many iterations are needed, the amount of work needed to reduce the residual by some fixed amount may be more for GMRES due to the increasing cost of the GMRES iterates. We will now describe the BCG algorithm:

Algorithm 6.1 *BCG - Bi-conjugate gradients*

1. **Start:** Choose x_0. Set $q_0 = r_0 = b - Ax_0$.
   Choose $\hat{r}_0$, $\hat{r}_0 \neq 0$, and set $\hat{q}_0 = \hat{r}_0$, $\rho_0 = \hat{r}_0^T r_0$.

2. **Iterate:** For $j = 1, 2, \ldots$ until satisfied do:

$$\begin{aligned}
\sigma_{j-1} &= \hat{q}_{k-1}^T A q_{k-1}, \\
\alpha_{j-1} &= \rho_{k-1}/\sigma_{k-1}, \\
x_k &= x_{k-1} + \alpha_{k-1} q_{k-1}, \\
r_k &= r_{k-1} - \alpha_{k-1} A q_{k-1}, \\
\hat{r}_k &= \hat{r}_{k-1} - \alpha_{k-1} A^T \hat{q}_{k-1},
\end{aligned}$$

   and

$$\begin{aligned}
\rho_k &= \hat{r}_k^T r_k, \\
\beta_k &= \rho_k/\rho_{k-1}, \\
q_k &= r_k + \beta_k q_{k-1}, \\
\hat{q}_k &= \hat{r}_k + \beta_k \hat{q}_{k-1}.
\end{aligned}$$

   If $r_k = 0$ or $\hat{r}_k = 0$, stop.

From this, we can see how the iterates are formed using only short recurrence relationships, whereas the GMRES algorithm (Algorithm 2.2) requires an increasing amount of

work as j increases. However, unlike GMRES, BCG iterates do not satisfy any optimality property such as (2.6). Indeed, BCG iterates can suffer wild oscillations in the norm of the residual, $\|r_k\|_2$, or more seriously, the algorithm may break down completely. Examining step 2 of Algorithm 6.1, then we see that BCG will fail at the kth step if

$$\hat{q}_{k-1}^T A q_{k-1} = 0 \text{ when } \hat{r}_{k-1} \neq 0,\ r_{k-1} \neq 0, \tag{6.15}$$

or if

$$\hat{r}_{k-1}^T r_{k-1} = 0 \text{ when } \hat{r}_{k-1} \neq 0,\ r_{k-1} \neq 0. \tag{6.16}$$

The first of these breakdowns (6.15) occurs when there exists no BCG iterate that satisfies the Galerkin condition (6.12). The second type of breakdown is due to failure of the underlying non-symmetric Lanczos algorithm, which can have a serious breakdown. As well as (6.15) and (6.16) being satisfied exactly, the use of finite-precision arithmetic means that numerical instability can be caused by very small values on the right-hand sides of the equations. Due to the possibility of breakdowns, BCG does not possess a finite convergence property such as Theorem 2.1 for GMRES.

Due to these reasons, BCG is not a widely used method, and alternatives were sought which preserved the low work and storage requirements of BCG but improving the robustness of the algorithm.

6.3.3 CGS and Bi-CGSTAB

One such method is the Conjugate Gradient Squared (CGS) method proposed in 1989 by Sonneveld [62], whose iterates satisfy the relation

$$r_k = \hat{p}_k(A)^2 r_0$$

where $\hat{p}_k$ is the BCG polynomial from (6.13). Note that the power of two on the polynomial gives the algorithm its name. Nachtigal, Reddy and Trefethen [49] state that CGS typically converges (or diverges) faster than BCG by a factor between 1 and 2. Now $\hat{p}_k^2 \in \mathcal{P}_{2k}$, so we can again use Lemma 2.1 to compare GMRES and CGS

residuals to obtain

$$\|r_{2k}^{\mathrm{GMRES}}\|_2 \leq \|r_k^{\mathrm{CGS}}\|_2.$$

By a rearrangement of Algorithm 6.1, CGS eliminates the need for the action of A^T on a vector, and becomes reliant merely on the action of A on a vector. Whilst this is not an issue for SPEEDUP problems where the transpose is freely available, it makes CGS more attractive for other applications where this is not the case.

Whilst the CGS method converges faster than BCG, it is still susceptible to breakdowns for the same reasons as BCG, and still possesses erratic convergence behaviour in the residual norm. The Bi-CGSTAB algorithm proposed by Van der Vorst in 1992 [67] has been shown to have the fast convergence properties of CGS but with a smoothly converging residual norm. Van der Vorst observed in [67] that although the oscillations of the CGS residual norm do not seem to affect the overall convergence of CGS, this is not necessarily the case. The practical effect in some situations is that large local peaks in the convergence curve result in cancellation caused by large corrections to the current iterate, resulting in loss of accuracy. Results in [67] show that in these cases, BiCGSTAB is much more efficient than CGS. However, it shares the failing of both BCG and CGS that breakdowns are possible. Nachtigal, Reddy and Trefethen construct a 2 by 2 example on which BCG, CGS and BiCGSTAB will break down, namely the skew-symmetric matrix

$$S = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix},$$

if the initial vector r_0 is real, and assuming $\hat{r}_0 = r_0$. However, in their discussion they comment that even given the possibilities for breakdown, it is remarkable how frequently these methods turn out to be effective. It is the BiCGSTAB method that we will use to examine the effectiveness of this type of method on SPEEDUP problems in the following set of numerical examples.

6.3.4 Numerical Experiments

We will now apply BiCGSTAB preconditioned by an exact LU factorisation of the Jacobian J_1 to matrices from Chapter 3, Example 3. The results are shown in Table 6.3.

If we compare these to the GMRES results for the same matrices shown in Table 3.1, then we can see that for the majority of the Jacobians, BiCGSTAB converges to the required tolerance in a similar number of iterations to the GMRES algorithm. Two points of interest are raised by these results. The first is that BiCGSTAB appears to have greater difficulty in producing an accurate solution for matrix J_5 than GMRES, requiring 102 iterations as opposed to the 36 required by GMRES. The convergence curve for this example is shown in Figure 6-2, and it shows the oscillatory behaviour typical of BCG-type methods. The GMRES convergence curve for this example is also shown on this figure for comparison. Since the convergence behaviour of BCG-type methods is not nearly as clearly understood as that of GMRES or CG(N), it is not possible to explain why BiCGSTAB should take many more iterations than GMRES. The second point of note is that BiCGSTAB yields a much more accurate approximate solution than GMRES for the example matrix J_6. Again, due to the lack of convergence theory, an explanation for such behaviour is not possible, but due to (2.45), it is possible that the BiCGSTAB residual is deficient in eigendirections corresponding to small eigenvalues of the preconditioned systems.

l	k	$\|e_k\|_2$
2	2	7.58×10^{-6}
3	17	2.67×10^{-7}
4	7	1.76×10^{-7}
5	102	5.80×10^{-8}
6	4	1.57×10^{-12}

Table 6.3: *Results of using $M_1 = J_1$ as a preconditioner for BiCGSTAB applied to Jacobian matrices from Example 3. Table shows the value of k required for BiCGSTAB residual norm to be less than 10^{-8}, and the error norm given by the kth iterate.*

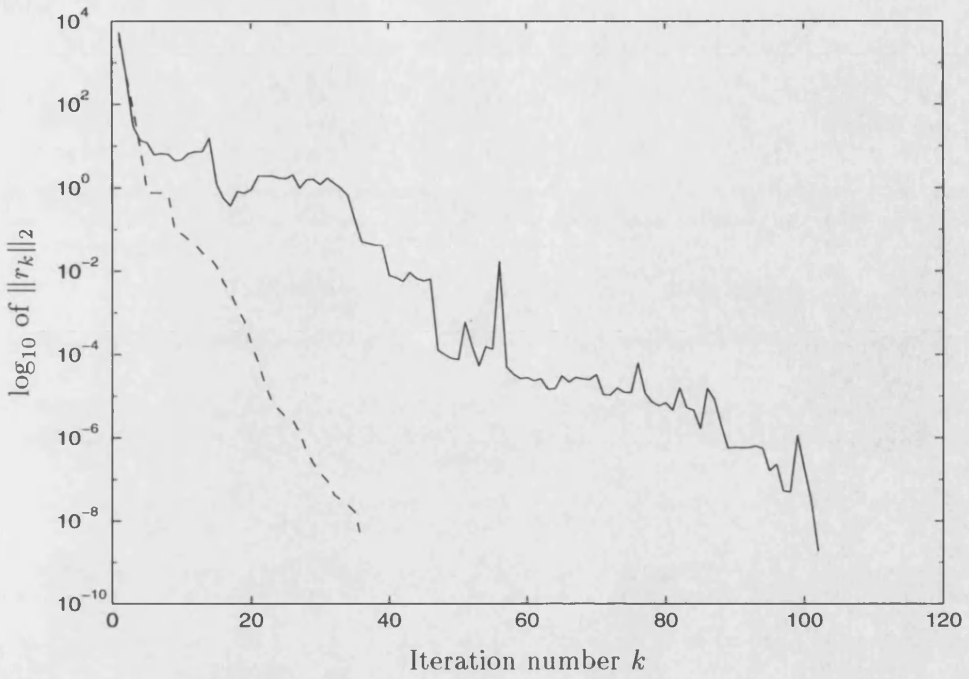

Figure 6-2: *Convergence curves for BiCGSTAB (solid line) and GMRES (dashed line) applied to matrix J_5 preconditioned with an exact inverse of matrix J_1 from example three, Chapter 3.*

We can use BiCGSTAB as an alternative iterative solver to GMRES in SPEEDUP integration runs. By a simple modification of algorithm 4.1, we obtain the following algorithm:

Algorithm 6.2 *FLUBiCG*

1. **Start:** Calculate $M^{-1} = J_1^{-1}$. Set $\delta x = -M^{-1} f(x)$

2. **Iterate:** For $l = 2, 3, \ldots$

   Apply BiCGSTAB, subject to a maximum of m iterations, to

   $$M^{-1} J_l \delta x = -M^{-1} f(x).$$

   If converged then set

   $$x \leftarrow x + \delta x, \; l = l + 1,$$

   else calculate new preconditioner:

   $$M^{-1} = J_l^{-1}.$$

   Goto 2.

This is essentially the same as the FLUGMR algorithm from Chapter 4, except there is no need to restart the BiCGSTAB iteration. This only leaves the decision of how many iterations to run BiCGSTAB for.

The impact of the growing cost of standard (non-restarted) GMRES was seen to be negligible in Chapter 4, due to the optimal value for the subspace length m being small (m=5 or 6). We may also recall that profiling revealed that the SOLVE phase of the preconditioner required the largest amount of CPU time when this optimal value was used. Thus it may be expected that BiCGSTAB, which requires twice as many preconditioned matrix-vector multiplications per iteration as GMRES, may not be able to compete with GMRES.

As an initial example, we will return to the BTX problem first presented in Chapter 4. This poses no real-time issues but provides a good basis for examining the relative performances of preconditioned GMRES and BiCGSTAB. As for GMRES, we will attempt to determine the optimal number of iterations to allow before we calculate a new preconditioner. The results for varying values of subspace length m are shown in Table 6.4.

As we can see, for comparable subspace lengths, BiCGSTAB is slower than GMRES.

m	CPU Time	SOLVE	FASTFAC
1	76.25	8390	439
2	87.31	10767	404
3	97.14	13117	395
4	107.12	15467	389
5	117.48	17817	379
6	66.86	8675	376
7	69.88	9452	365
8	72.43	10133	356
Optimal GMRES			
5,1	84.80	8744	357
Direct Method			
	43.9	812	812

Table 6.4: *Results of BiCGSTAB as an iterative solver in SPEEDUP for the BTX problem. All runs 1* FASTFAC *and 1* ANALYSE *call. Shown for comparison are values for the optimal GMRES method and the direct method.*

Unlike GMRES, the fastest run was with a subspace of length 1 (from Chapter 4, we saw that the optimal value of m for GMRES was generally 5 or 6). This implies that, for this example at least, the most effective BiCGSTAB is one with subspace length 0, i.e. just the direct solver.

When we apply BiCGSTAB to a more serious example from Chapter 4, we find that a breakdown occurs during the integration with (6.16). Whilst this breakdown is detectable and can be worked around, it results in having to use the direct solver to continue the integration. If breakdowns occur regularly, then this may require many additional factorisations, which is what we are trying to avoid by using an iterative solver in the first instance. In view of this, and the inferior performance of BiCGSTAB relative to GMRES indicated by the BTX example, we can conclude that BiCGSTAB is not as suited to use in SPEEDUP integration as GMRES. Figure 6-3 shows the CPU-time/real-time ratio for BiCGSTAB with $m = 5$ for the portion of the integration prior to the first breakdown, compared to that of the optimal GMRES. The ratio for BiCGSTAB is consistently higher than that of the GMRES algorithm.

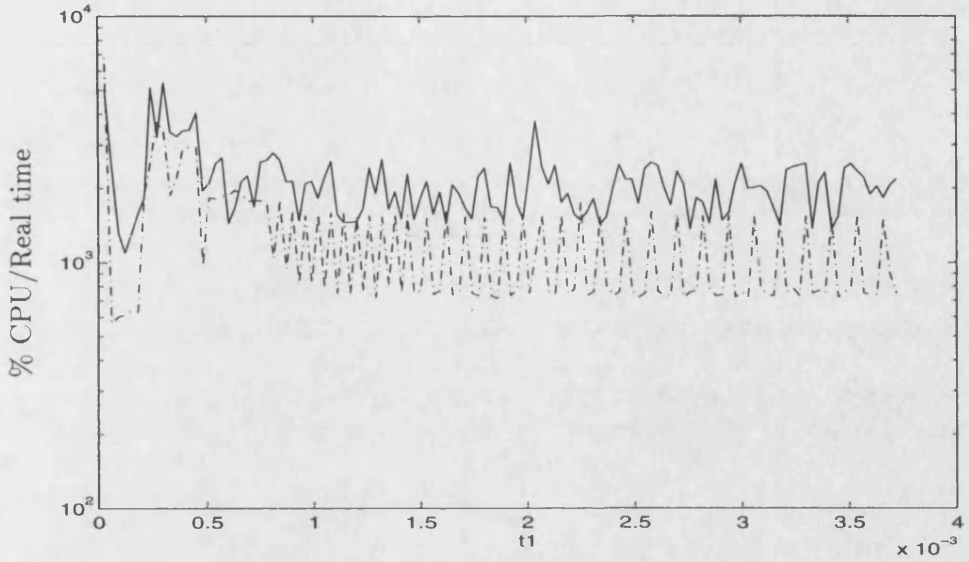

Figure 6-3: *CPU to real-time ratio for initial part of Example 2 integration. The solid line is FLUBiCG(5), and the dot-dashed line is FLUGMR(5,0).*

6.4 QMR: Quasi-Minimal residuals

6.4.1 Look-ahead Lanczos

The breakdown of the Lanczos method can be avoided by using a 'look-ahead' version [54], whereby potential breakdowns are detected in advance, and the biorthogonality condition (6.8) is relaxed to allow the process to continue. The resulting matrix $H_k^{\text{lookahead}}$ is block-tridiagonal, as opposed to the tridiagonal matrix H_k,(6.10), generated by the standard nonsymmetric Lanczos algorithm. This process still generates bases for the two Krylov subspaces $\mathcal{K}_k(A, v_1)$ and $\mathcal{K}_k(A^T, w_1)$, and the basis vectors $V_k = [v_1 \dots v_k]$ and $W_k = [w_1 \dots w_k]$ satisfy

$$\begin{aligned} AV_k &= V_k H_k^{\text{lookahead}}, \\ A^T W_k &= W_k H_k^{\text{lookahead}^T}, \end{aligned} \tag{6.17}$$

which should be compared with (2.9) and (6.11).

6.4.2 The QMR method

In 1991, Freund and Nachtigal proposed the Quasi-minimal residual method (QMR) for solving nonsymmetric linear systems [29]. This method is based on the look-ahead variant of the Lanczos algorithm, thus addressing the problem of potential breakdown in BCG-type methods caused by failures in the standard Lanczos process. Its iterates are still formed from short-term recurrence relationships, but its iterates have the attractive property that they satisfy a quasi-optimality property. Recalling the GMRES algorithm (§2.3.3), the minimisation of (2.10) was changed from an $n \times k$ least-squares problem to a $(k+1) \times k$ one (2.14) by taking advantage of the fact that the matrices A, V_k, V_{k+1} and $\bar{H}_k$ satisfy the relation

$$AV_k = V_{k+1}\bar{H}_k$$

and that V_{k+1} is l_2-orthonormal. For the look-ahead Lanczos process, we can write a similar relation, i.e.

$$AV_k = V_{k+1}\bar{H}_k^{\text{lookahead}}, \tag{6.18}$$

where $\bar{H}_k^{\text{lookahead}}$ is the same as $H_k^{\text{lookahead}}$ with the addition of an extra row whose only non-zero entry is $h_{k+1,k}$. Now since the QMR iterates satisfy

$$x_k \in x_0 + \mathcal{K}_k(A, r_0)$$

and the vectors V_k span $\mathcal{K}_k(A, r_0)$, we can write

$$x_k = x_0 + V_k y,\ y \in \mathbb{R}^n,$$

and from (6.18), we see that

$$r_k = r_0 - AV_k y = r_0 - V_{k+1}\bar{H}_k^{\text{lookahead}} y = V_{k+1}(\rho_0 e_1 - \bar{H}_k^{\text{lookahead}} y), \tag{6.19}$$

where $\rho_0 = \|r_0\|_2$. We would like the iterates to satisfy a condition such as

$$\|r_k\|_2 = \min_{y \in \mathbb{R}^k} \|b - AV_k y\|_2.$$

From (6.19), we have

$$\|r_k\|_2 = \|V_{k+1}(\rho_0 e_1 - \bar{H}_k^{\text{lookahead}} y)\|_2. \tag{6.20}$$

Unfortunately, the matrix V_{k+1} is not l_2-orthonormal in general, so we cannot satisfy the optimality property without solving an $n \times k$ minimisation property. Instead, we choose iterates that satisfy the quasi-optimality property

$$\|\rho_0 e_1 - \bar{H}_k^{\text{lookahead}} y_k\|_2 = \min_{y \in \mathbb{R}^k} \|\rho_0 e_1 - \bar{H}_k^{\text{lookahead}} y\|_2 \tag{6.21}$$

obtained by discarding V_{k+1} from (6.20) anyway. Thus the residuals from QMR are said to be *quasi-minimal.*

6.4.3 Numerical Experiments

As for the BiCGSTAB algorithm, we will first present results for QMR preconditioned with the exact inverse of a previous Jacobian matrix applied to an example from Chapter 3, followed by the application of QMR with this preconditioning to a SPEEDUP integration run from an example from Chapter 4.

l	k	$\|e_k\|_2$
2	3	2.53×10^{-6}
3	33	1.72×10^{-8}
4	8	3.16×10^{-11}
5	78	8.01×10^{-8}
6	5	6.51×10^{-10}

Table 6.5: *Results of using $M_1 = J_1$ as a preconditioner for QMR applied to Jacobian matrices from Example 3. Table shows the value of k required for QMR residual norm to be less than 10^{-8}, and the error norm given by the kth iterate.*

Again, we may expect QMR to be more expensive than GMRES when we consider the fact that the optimal value of m in the FLUGMR algorithm from Chapter 4 is small, resulting in little impact on the solution time of the increasing cost of GMRES iterates. This is again because two preconditioned matrix-vector multiplications are required

per QMR iteration, doubling the amount of SOLVE calls which already dominate the solution time.

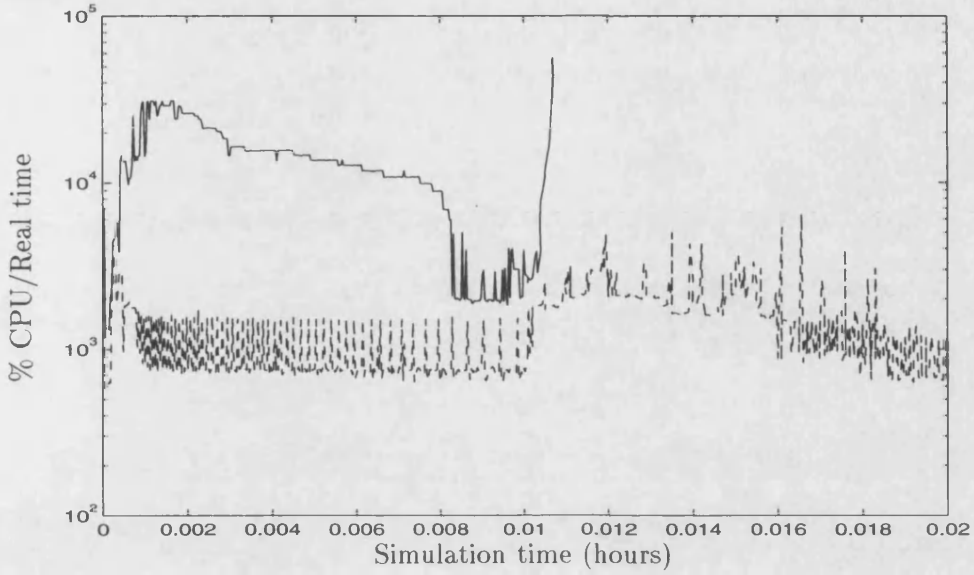

Figure 6-4: *CPU-time to real-time ratio (solid line) for FLUQMR applied to Example 2 from Chapter 4. Breakdown occurs at time t=0.0107. The dashed line is for FLUGMR(5,0).*

If we apply the FLUQMR algorithm, derived in the same manner as FLUBiCG, to example 2 from Chapter 4, a severe problem occurs. We have seen in §6.4.2 that the basis vectors generated for $\mathcal{K}_k(A)$ and $\mathcal{K}_k(A^T)$ are not normalised. With 'real-world' flowsheets, this becomes a handicap as the ill-conditioning of the preconditioned system causes the vectors to exhibit enormous growth, resulting in extremely slow convergence, and eventual overflow at some stage of the iteration. Figure 6-4 shows the CPU-time to real-time ratio for FLUQMR applied to Example 2 from Chapter 4, including the breakdown at time $t = 0.0107$, compared to that for the optimal FLUGMR solution. This breakdown is not recoverable from, and would cause additional arithmetic to prevent and correct. In the original presentation of QMR [29], Freund and Nachtigal suggest using a weight matrix $\Omega^{(n)} = \text{diag}(\omega_1, \omega_2, \ldots, \omega_{n+1})$ where the ω_j are greater than zero, to modify the scaling of the problem.

This approach could be used here, but an alternative approach was used. It was found that by insisting that FACTORISE calls are made instead of FASTFAC calls, the effects

of the ill conditioning could be reduced substantially. The calculation of a new pivot sequence instead of using an existing one produces a linear system sufficiently well-conditioned to prevent the errors outlined above from occurring. The results are shown

m	Its	SOLVE	FAC	ANA	CPU Time
1	1830	12612	1647	3	1838.17
2	1827	17568	847	3	1798.07
3	1829	22649	652	5	1773.07
4	1832	25550	429	5	1560.59
5	1830	29452	357	5	1680.28
6	1830	32699	316	4	1717.20
Optimal GMRES					
6,0	1828	12749	21	7	959.06
MA28, FASTNEWTON					
	3442	3442	19	19	763.78

Table 6.6: *Results of QMR as an iterative solver in SPEEDUP for Example 2. Table shows number of nonlinear iterations,* SOLVE, FACTORISE *and* ANALYSE *calls, and total CPU time for the integration for various values of m. Shown for comparison are values for the optimal GMRES and direct methods.*

in Table 6.6. As we can see, the cost of the additional work required by the FACTORISE calls over the cheaper FASTFAC calls, and the additional SOLVE calls mean that performance is significantly slower than both the optimal GMRES solver and the optimal direct method, in this case MA28 with the FASTNEWTON option. Unlike the BiCGSTAB method, which had a 'best' subspace length of $m = 1$ (ie the best result was not to use BiCGSTAB at all), QMR has a best value of $m = 4$.

6.5 Conclusions

In this chapter, we have examined three alternative iterative solvers to GMRES. All three were shown to have deficiencies in performance when applied to the examples in this thesis.

The CGN method suffered from extremely slow convergence, meaning that implementing it as a SPEEDUP solver would be a futile exercise. This result was not surprising, given the 'real world' nature of the Jacobian matrices we are dealing with.

Both the BiCGSTAB and QMR methods may have been expected to perform well, despite the lack of convergence results concerning these algorithms. However, they proved slower than GMRES despite showing comparable rates of convergence on the test matrices. This is due to the fact that our GMRES implementation was found to produce optimal performance with quite short subspaces, ($m = 5$ or 6), and the impact of needing twice as many preconditioned matrix-vector multiplications weighed against them.

Finally, both BiCGSTAB and QMR proved unsuitable for use in SPEEDUP integrations due to a lack of robustness. BiCGSTAB suffered a breakdown at some stage of an integration. Although it would be possible to bypass such an occurrence and continue the integration, if this were to occur frequently, then performance would be severely impaired.

QMR suffered a similar breakdown, but this has more to do with the ill-conditioned nature of the Jacobian matrices than the QMR algorithm. The fact that the basis vectors for the two biorthogonal subspaces are not scaled or orthogonalised leads to enormous growth in the basis vectors, resulting in overflow and breakdown. By constructing a more stable preconditioner, the conditioning was improved sufficiently to allow solution. However, the additional work this entails, coupled with the extra matrix-vector multiplications required by QMR result in much slower than GMRES.

Thus we can conclude that GMRES is the most suitable of the four iterative methods for solving the linear systems that occur from the integration of SPEEDUP flowsheets. The QMR breakdown emphasises the extreme degree of ill-conditioning present in the Jacobian matrices that arise from the nonlinear systems that we are solving.

Chapter 7

Final Conclusions and Future Work

We can summarise the achievements of this thesis as follows: We have developed a robust iterative solver, FLUGMR, for a series of linear systems (1.1), based on the GMRES method [61], for use in SPEEDUP. These linear systems arise from the time integration of systems of DAE's (1.2), and are typically severely ill-conditioned. The DAE systems result from the modelling of real-world process engineering problems, and can contain tens of thousands of equations. FLUGMR uses a full LU factorisation of an initial matrix to precondition GMRES applied to subsequent linear systems.

Using a novel approach to the GMRES error analysis, we were able to obtain convergence rate estimates related to the distribution of the spectra of a matrix, and showed that if the majority of the eigenvalues were suitably clustered, the rate of convergence was unaffected by large or small outliers. Numerical experiments on several example matrices from SPEEDUP flowsheets showed that conventional ILU preconditioning methods were ineffective. Examining the pseudospectra of the upper Hessenberg matrix arising from the GMRES process showed that the spectra of the ILU preconditioned systems were subject to a high degree of non-normality, thus explaining, by appeal to the convergence bounds and other indicators, the poor performance of this preconditioner. Experiments using an exact LU factorisation of a previous matrix from the

sequence as a preconditioner yielded much better results, which were again justified by examining the pseudo spectra of the upper Hessenberg matrix, and by direct appeal to the convergence bounds we have derived.

The performance of FLUGMR was tested on a number of real-world SPEEDUP flowsheets modelling actual process engineering applications. The solver was found to be robust, enabling SPEEDUP to complete the integration of all the flowsheets used to test the algorithm. However, its performance was not as good as the optimal combination of a direct solver coupled with an approximate Newton method. Whilst fewer factorisations were required than for the direct solver, the large increase in the number of SOLVE operations required by FLUGMR over the direct solver resulted in slower performance.

Inexact Newton methods were considered, and for some choices of the forcing term performance was better than the black-box solver. However, any performance gains were negligable, and the more sophisticated choices of forcing term actually resulted in often drastically slower performance. This is probably caused by poor agreement with the preconditioned GMRES residual norm and the actual achieved error norm. This indicates that care should be exercised when applying theoretical results to practical applications. Alternative Krylov methods were tried as an alternative to GMRES, but were found to be slower and lacked robustness.

In conclusion, we can state that the state of the art iterative methods for general linear systems are not able to compete with the best optimised direct solution methods used in conjunction with an approximate Newton method, when applied to the real time integration of realistic process modelling applications. The root cause of the problem lies in the ill-conditioning of the matrices we are faced with. Such ill-conditioning means popular sparse matrix preconditioners like ILU are ineffective, producing slow and inaccurate convergence for GMRES. To obtain satisfactory convergence, it is neccessary to use a direct method as a preconditioner. The fill-in that occurs during the factorisation means that the application of the preconditioner is the most expensive part of the solution process.

From the work carried out with the other Krylov subspace-based iterative solvers in Chapter 6, it is clear that no iterative method exists at the moment that will compete

with the direct method/approximate Newton strategy. If a method was devised that could take advantage of the nature of the problem we are dealing with, then this could offer a potential improvement. Since SPEEDUP flowsheets vary from one to the next, how such a method could be devised is not clear. At the least, it would probably entail integrating the solver with the block decomposition technique described in §1.4.2.

Perhaps a larger benefit may be obtained by trying to improve the conditioning of the linear systems. Whether this is achieved by some pre-processing measure or an improvement in the scaling of the models offered for solution is undecided. This approach is not related to the method used to solve the resulting linear systems, however, and is therefore beyond the scope of any related work. Performance can always be improved with the introduction of faster hardware, and in the short term perhaps this is the only guaranteed way of improving the real-time performance of the real-world problems we have seen in this thesis.

Bibliography

[1] W. E. Arnoldi. The principle of minimized iteration in the solution of the matrix eigenvalue problem. *Quart. Appl. Math.*, 9:17–29, 1951.

[2] Aspen Technology, Inc. *SPEEDUP User Manual.*

[3] O. Axelsson. *Iterative Solution Methods.* CUP, 1994.

[4] O. Axelsson and V.A. Barker. *Finite element solution of boundary value problems - theory and computation.* Academic Press, 1984.

[5] O. Axelsson and G. Lindskog. On the eigenvalue distribution of a class of preconditioning methods. *Numer. Math.*, 48:499–523, 1986.

[6] O. Axelsson and G. Lindskog. On the rate of convergence of the preconditioned conjugate gradient method. *Numer. Math.*, 48:499–523, 1986.

[7] O. Axelsson and N. Munksgaard. Analysis of incomplete factorisations with fixed storage allocation. In D. J. Evans, editor, *Preconditioning methods, Theory and applications*, pages 219–241. Gordon and Breach, London, 1983.

[8] A Bjork. Solving linear least-squares problems by Gramm-Schmidt orthogonalisation. *BIT*, 7:1–21, 1967.

[9] P. N. Brown and Y. Saad. Hybrid Krylov methods for nonlinear systems of equations. *SIAM J. Sci. Stat. Comput.*, 11:450–481, May 1990.

[10] P.N. Brown and A.C. Hindmarsh. Matrix-free methods for stiff systems of ODE's. *SIAM J. Numer. Anal.*, 23:610–688, 1986.

[11] J.J. Buoni. Incomplete factorizations of singular M-matrices. *SIAM J. Alg. Disc. Meth.*, 7:193–198, 1986.

[12] G. D. Byrne and P. R. Ponzi. Differential-algebraic systems, their applications and solutions. *Comput. Chem. Engng.*, 12:377–382, 1988.

[13] X.-C. Cai, D. Gropp, D. E. Keyes, and M. D. Tidriri. Newton-Krylov-Schwarz methods in CFD. In R. Rannacher, editor, *Proceedings of the International Workshop on the Navier-Stokes Equations*, Notes in Numerical Fluid Mechanics. Braunschweig, Vieweg Verlag, 1994.

[14] F. Chatelin. *Eigenvalues of Matrices.* John Wiley & Sons, 1993.

[15] J. W. Daniel. The conjugate gradient method for linear and nonlinear operator equations. *SIAM J. Numer. Anal.*, 4:10–26, 1967.

[16] R. S. Dembo and T. Steihaug. Truncated Newton algorithms for large scale optimization. *Math. Prog.*, 26:190–212, 1983.

[17] R.S. Dembo, S.C. Eisenstat, and T. Steihaug. Inexact Newton methods. *SIAM J. Numerl. Anal.*, 19:400–408, 1982.

[18] J. E. Dennis, Jr. and R. B. Schnabel. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations.* Prentice-Hall Series in Computational Mathematics. Prentice-Hall, 1983.

[19] I. S. Duff. MA28 - a set of FORTRAN subroutines for sparse unsymmetric linear equations. Technical Report HL80/3459, AERE Harwell, November 1980.

[20] I. S. Duff. On algorithms for obtaining a maximum transversal. *ACM Trans. Math. Softw.*, 7(3):315–330, 1981.

[21] I. S. Duff. Permutations for a zero-free diagonal. *ACM Trans. Math. Softw.*, 7:387–390, 1981.

[22] I. S. Duff, A. M. Erisman, and J. K. Reid. *Direct Methods for Sparse Matrices.* Monographs on Numerical Analysis. Oxford Science Publications, 1986.

[23] I. S. Duff and J. K. Reid. *MA48, a FORTRAN code for direct solution of sparse unsymmetric linear systems of equations*, September 1993.

[24] T. Dupont, R. P. Kendall, and H. H. Rachford. An approximate factorisation procedure for solving self-adjoint elliptic difference equations. *SIAM J. Numer. Anal.*, 5:559–573, 1968.

[25] S. C. Eisenstat and H. F. Walker. Globally converegent inexact Newton methods. *SIAM J. Optimization*, 4:393–422, 1994.

[26] S. C. Eisenstat and H. F. Walker. Choosing the forcing term in an inexact Newton method. *SIAM J. Sci. Comput.*, 17:16–32, 1996.

[27] H.C. Elman. A stability analysis of incomplete LU factorisations. *Math. Comp.*, 47(175):191–217, 1986.

[28] R.W. Freund, G.H.Golub, and N.M.Nachtigal. Iterative solution of linear systems. In *Acta Numerica*. CUP, 1991.

[29] R.W. Freund and N.M. Nachtigal. QMR: a quasi-minimal residual method for non-Hermitian linear systems. *Numer. Math.*, 60:315–339, 1991.

[30] C. W. Gear. The simultaneous numerical solution of differential-algebraic equation. *IEEE Trans. Circuit Theory*, CT-18:89–95, 1971.

[31] G. H. Golub and C. F. Van Loan. *Matrix Computations*. John Hopkins University Press, second edition, 1989.

[32] A. Greenbaum, V. Pták, and Z. Strakoš. Any nonincreasing convergence curve is possible for GMRES. *SIAM J. Matrix Anal. Appl.*, 17(3):465–469, July 1996.

[33] I. Gustafsson. A class of first order factorisations. *BIT*, 18:142–156, 1978.

[34] W. Hackbusch. *Iterative solution of large sparse systems of equations*. Applied Mathematical Sciences. Springer-Verlag, 1994.

[35] M.R. Hestenes and E. Stiefel. Methods of conjugate gradients for solving linear systems. *J. Res. Nat. Bur. Standards*, 49:409–436, 1952.

[36] Y. Huang and H.A. van der Vorst. Some observations on the convergence behaviour of GMRES. Technical Report 89-09, Delft University of Technology, 1989.

[37] C. Johnson. *Numerical solution of partial differential equations by the finite element method*. CUP, 1987.

[38] C. T. Kelley. *Iterative Methods for Linear and Nonlinear Equations.* Number 16 in Frontiers in applied mathematics. SIAM, 1995.

[39] D.S. Kershaw. The incomplete Cholesky-conjugate gradient method for the itertive solution of systems of linear equations. *J. Comput. Phys.*, 26:43–65, 1978.

[40] C. Lanczos. An iterative method for the solution of the eigenvalue problem of linear differential and integral operators. *J. Res. Natl Bur. Stand.*, 45:255–282, 1950.

[41] C. Lanczos. Solution of systems of linear equations by minimized iterations. *J. Res. Natl Bur. Stand.*, 49:33–53, 1952.

[42] J.R. Leis and M. A. Kramer. Sensitivity analysis of systems of differential and algebraic equations. *Comput. & Chem. Engng.*, 9:93–96, 1985.

[43] P. Lötstedt and L. Petzold. Numerical solution of nonlinear differential equations with algebraic constraints I: Conergence results for backward differentiation formulas. *Math. Comp.*, 46:491–516, 1986.

[44] P. Lötstedt and L. Petzold. Numerical solution of nonlinear differential equations with algebraic constraints II: Practical implications. *SIAM J. Sci. Statist. Comput.*, 7:720–733, 1986.

[45] H. M. Markowitz. The elimination form of the inverse and its application to linear programming. *Management Sci.*, 3:255–269, 1957.

[46] R. März. Numerical methods for differential algebraic equations. In *Acta Numerica.* CUP, 1991.

[47] J.A. Meijerink and H.A. van der Vorst. An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix. *Math. Comp.*, 31(137):148–162, 1977.

[48] J.A. Meijerink and H.A. van der Vorst. Guidelines for the usage of incomplete decompositions in solving sets of linear equations as they occur in practical problems. *J. Comp. Phys.*, 44:134–155, 1981.

[49] N.M. Nachtigal, S.C. Reddy, and L.N. Trefethen. How fast are nonsymmetric matrix iterations? *SIAM J. Matrix Anal. Appl.*, 13:778–795, July 1992.

[50] Y. Notay. Incomplete factorizations of singular linear systems. *BIT*, 29:682–702, 1989.

[51] Y. Notay. On the robustness of modified incomplete factorization methods. *Intern. J. Computer Math*, 40:121–141, 1992.

[52] C. C. Pantelides. Speedup - recent advances in process simulation. *Comput. Chem. Eng.*, 12:745–755, 1988.

[53] B.N. Parlett. *The symmetric eigenvalue problem.* Prentice Hall, 1980.

[54] B.N. Parlett, D.R. Taylor, and Z.A. Liu. A look-ahead Lanczos algorithm for unsymmetric matrices. *Math. Comp.*, 44:105–124, 1985.

[55] J. K. Reid. A note on the stability of gaussian elimination. *J. Inst. Maths. Applics.*, 8:374–375, 1971.

[56] T.J. Rivlin. *The Chebyshev Polynomials.* John Wiley & Sons, 1990.

[57] Y. Saad. Krylov subspace methods for solving large unsymmetric linear systems. *Math. Comput.*, 37:105–126, 1981.

[58] Y. Saad. Practical use of some Krylov subspace methods for solving indefinite and unsymmetric linear systems. *SIAM J. Sci. Stat. Comput.*, 5:203–228, 1984.

[59] Y. Saad. *Numerical Methods for large eigenvalue problems.* John Wiley & Sons, 1992.

[60] Y. Saad. *SPARSKIT: A basic tool kit for sparse matrix computations*, 1994. Version 2.

[61] Y. Saad and M.H. Schultz. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM J. Sci. Stat. Comput.*, 7:856–869, July 1986.

[62] P. Sonneveld. CGS: A fast Lanczos-type solver for nonsymmetric linear systems. *SIAM J. Sci. Stat. Comput.*, 10:36–52, 1989.

[63] G. W. Stewart. The economical storage of plane rotations. *Numer. Math.*, 25:137–138, 1976.

[64] R. E. Tarjan. Depth-first search and linear graph algorithms. *SIAM J. Comput.*, 1:146–160, 1972.

[65] K.C. Toh and L.N. Trefethen. Calculation of pseudospectra by the Arnoldi iteration. *SIAM J. Sci. Comput.*, 17(1):1–15, 1996.

[66] L.N. Trefethen. Pseudospectra of matrices. In D.F. Griffiths and G.A. Watson, editors, *Numerical Analysis 1991*. Longman Scientific and Technical, 1992.

[67] H. A. Van der Vorst. Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems. *SIAM J. Sci. Stat. Comput.*, 13:631–644, 1992.

[68] H.A. van der Vorst and C. Vuik. The superlinear convergence of GMRES. *J. Comp. Appl. Math.*, 48:327–341, 1993.

[69] R.S. Varga, E.B. Saff, and V. Mehrmann. Incomplete factorisation matrices and connections with H-matrices. *SIAM J. Numer. Anal.*, 17:787–793, 1980.

[70] H. F. Walker. A GMRES-backtracking Newton iterative method. Technical Report 3/94/74, Dept. of Mathematics and Statistics, Utah State University, 1994.

[71] H.F. Walker. Implementation of the GMRES method using Householder transformations. *SIAM J. Sci. Stat. Comput.*, 9:152–163, 1988.

[72] J. H. Wilkinson. Error analysis of direct methods of matrix inversion. *J. ACM*, 8:281–330, 1961.